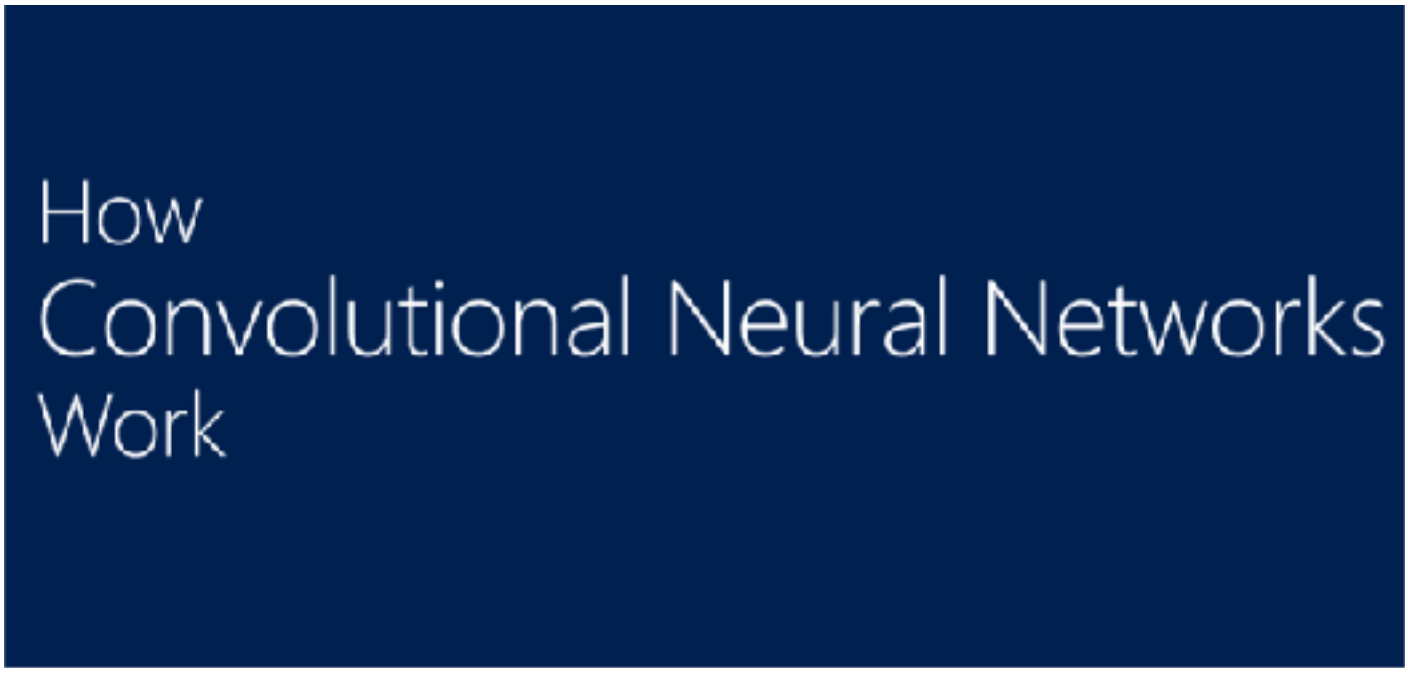
[透析] 卷积神经网络CNN究竟是怎样一步一步工作的?

zhwhong (/u/38cd2a8c425e) ＋关注
2016.12.17 23:55* 字数 3518 阅读 14363 评论 56 喜欢 211 赞赏 7
(/u/38cd2a8c425e)

- 视频地址：https://www.youtube.com/embed/FmpDIaiMIeA (https://link.jianshu.com?t=https://www.youtube.com/embed/FmpDIaiMIeA)
- 文档参阅：pdf [2MB] & ppt [6MB] & Web View (https://link.jianshu.com?t=http://brohrer.github.io/how_convolutional_neural_networks_work.html) & GitBook (https://link.jianshu.com?t=https://brohrer.mcknote.com/how_machine_learning_works/how_convolutional_neural_networks_work.html)
- 补充知识：深度学习 — 反向传播理论推导 (https://www.jianshu.com/p/408ab8177a53)



当你听到说深度学习打破了某项新技术障碍，那么十有八九就会涉及到卷积神经网络。它们也被称作CNNs或着ConvNets，是深层神经网络领域的主力。它们已经学会对图像进行分类，在某些情况下甚至超过了人类。如果有一个方法证明了这种假设，那就是CNN。特别酷的一点就是，当你将它们分解为基本模块时，它们很容易被理解。这里有一个视频 (https://link.jianshu.com?t=https://www.youtube.com/embed/FmpDIaiMIeA)，很详细地讨论了关于这些图像问题。

LeNet-5

Classfication

先验工作

- 【icml09 - Convolutional Deep Belief Networks.pdf (https://link.jianshu.com?t=http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf)】

Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations

Honglak Lee HLLEE@CS.STANFORD.EDU
Roger Grosse RGROSSE@CS.STANFORD.EDU
Rajesh Ranganath RAJESHR@CS.STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305, USA

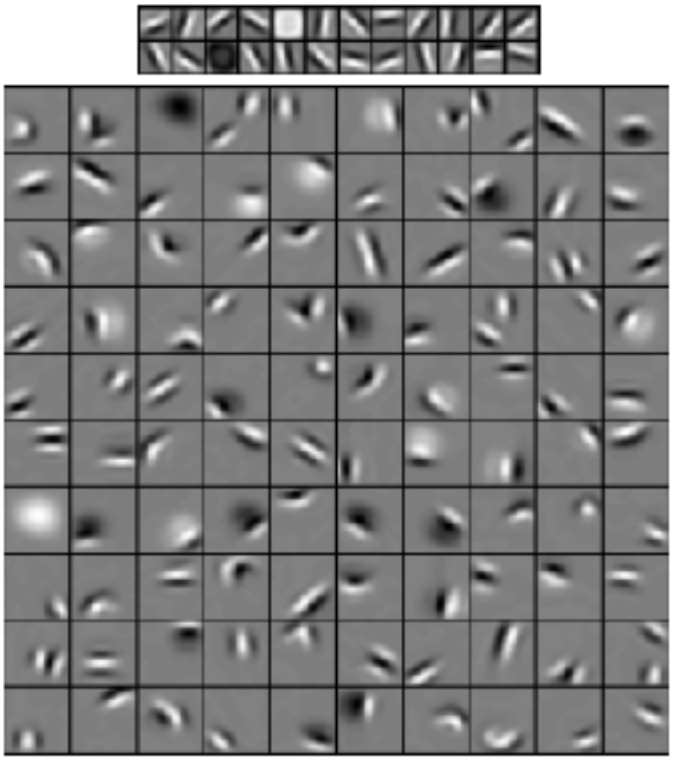


Figure 2. The first layer bases (top) and the second layer bases (bottom) learned from natural images. Each second layer basis (filter) was visualized as a weighted linear com-

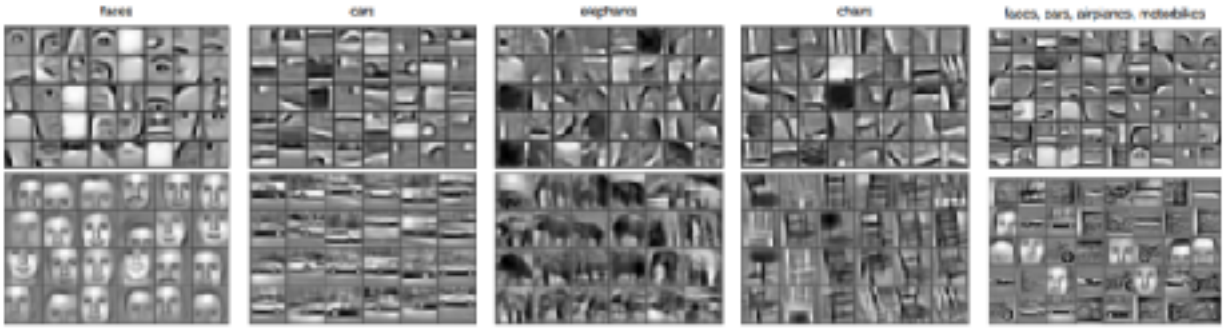
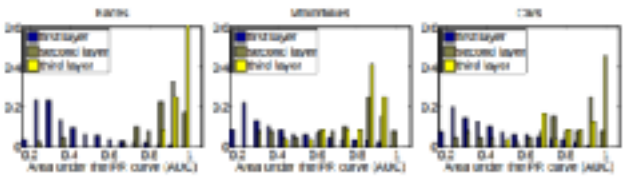


Figure 3. Columns 1-4: the second layer bases (top) and the third layer bases (bottom) learned from specific object categories. Column 5: the second layer bases (top) and the third layer bases (bottom) learned from a mixture of four object categories (faces, cars, airplanes, motorbikes).



Features	Faces	Motorbikes	Cars
First layer	0.39±0.17	0.44±0.21	0.43±0.19
Second layer	0.86±0.13	0.69±0.22	0.72±0.23
Third layer	0.95±0.03	0.81±0.13	0.87±0.15

- 【Playing Atari with Deep Reinforcement Learning (https://link.jianshu.com?t=http://arxiv.org/pdf/1312.5602v1.pdf)】

Playing Atari with Deep Reinforcement Learning

Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

- 【Robot Learning Manipulation Action Plans (https://link.jianshu.com?t=http://www.umiacs.umd.edu/%7Eyzyang/paper/YouCookMani_CameraReady.pdf)
】

Robot Learning Manipulation Action Plans by "Watching" Unconstrained Videos from the World Wide Web

Yezhou Yang
University of Maryland
yzyang@cs.umd.edu

Yi Li
NICTA, Australia
yi.li@nicta.com.au

Cornelia Fermüller
University of Maryland
fer@umiacs.umd.edu

Yiannis Aloimonos
University of Maryland
yiannis@cs.umd.edu

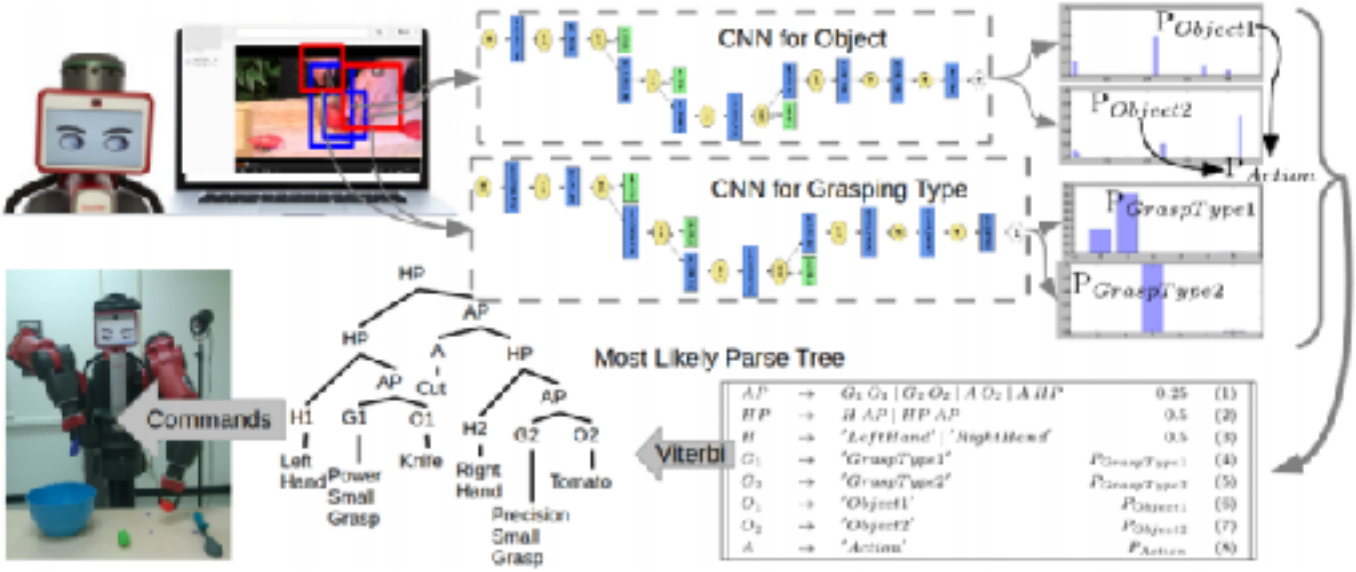
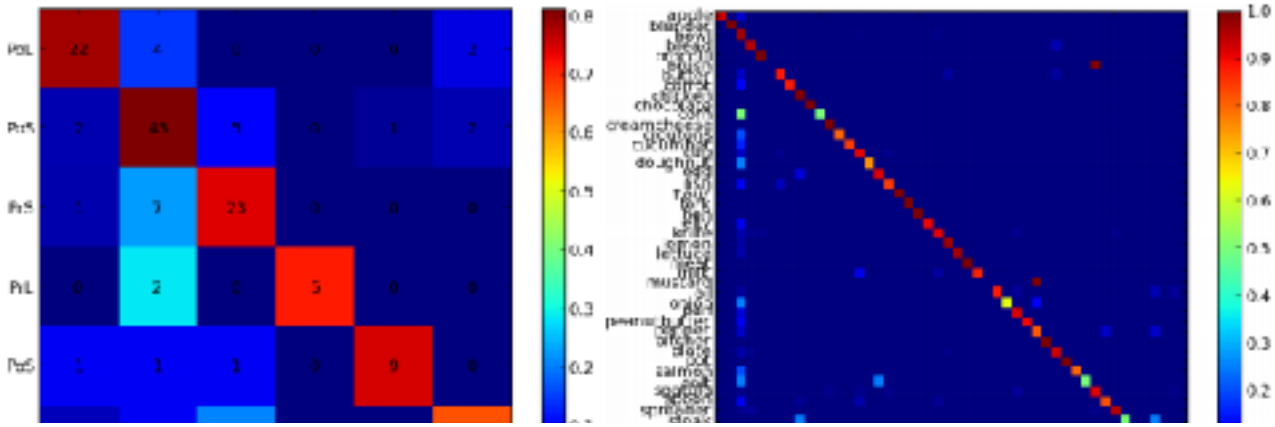


Figure 1: The integrated system reported in this work.



A toy ConvNet：X's and O's

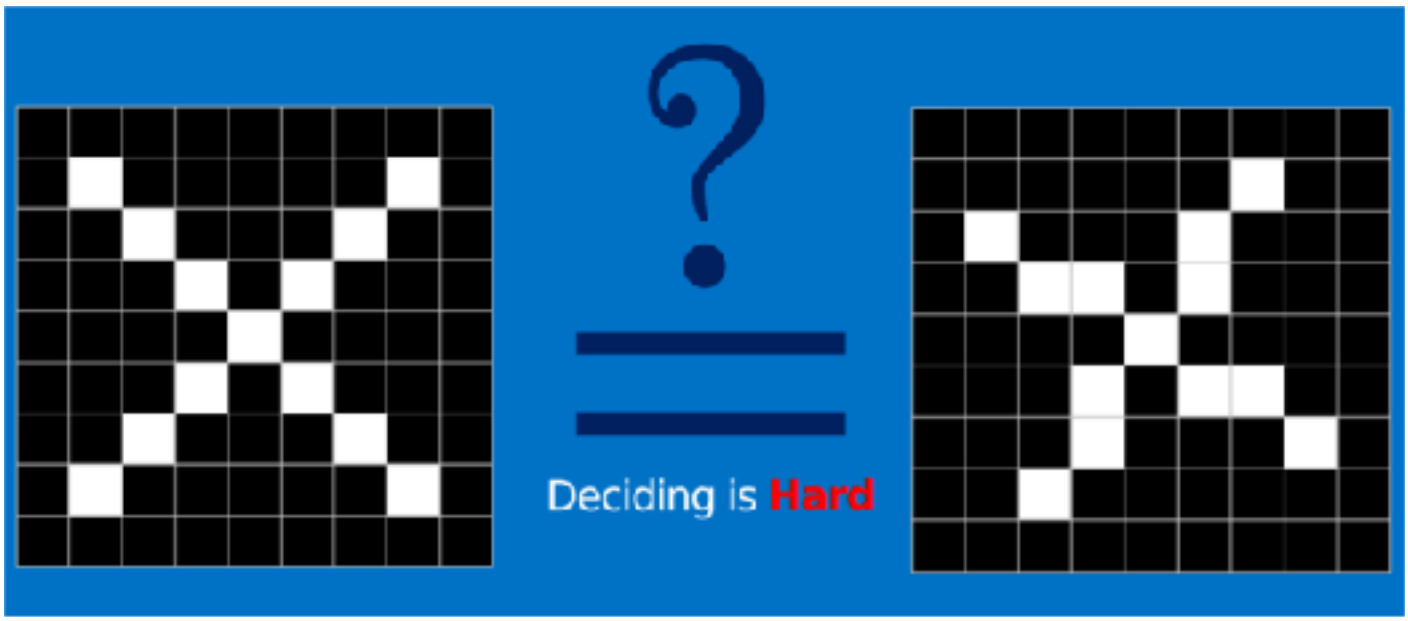
> **识别一幅图片是包含有字母"X"还是字母"O"?**

为了帮助指导你理解卷积神经网络，我们讲采用一个非常简化的例子：确定一幅图像是包含有"X"还是"O"?

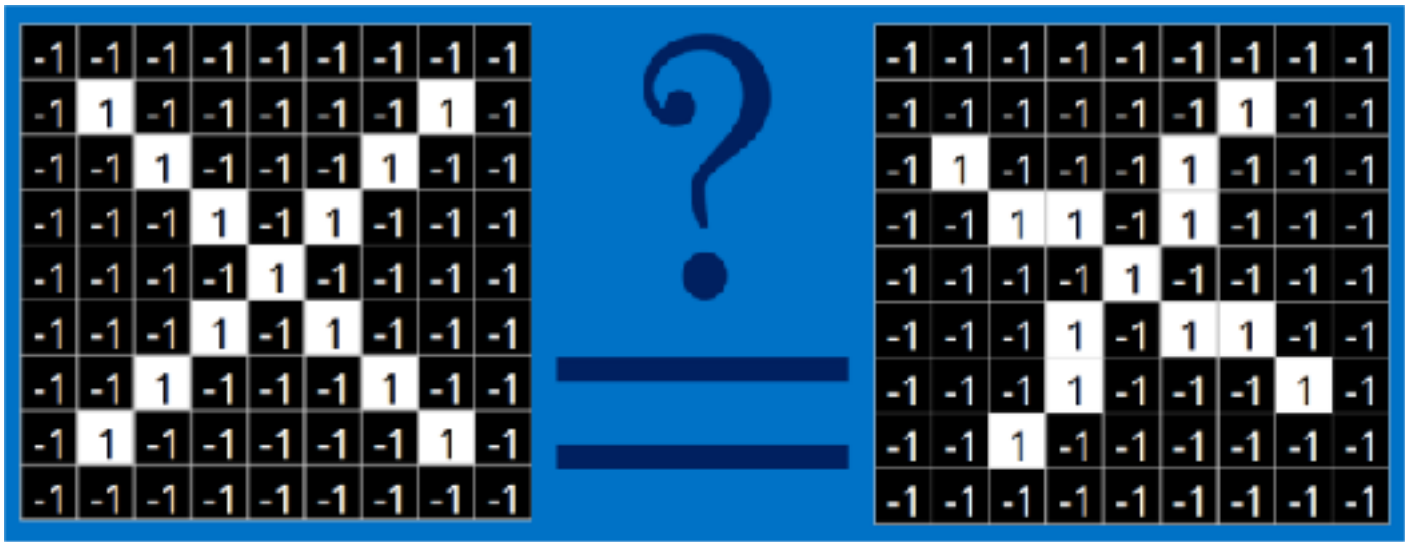
这个例子足够说明CNN背后的原理，同时它足够简单，能够避免陷入不必要的细节。在CNN中有这样一个问题，就是每次给你一张图，你需要判断它是否含有"X"或者"O"。并且假设必须两者选其一，不是"X"就是"O"。理想的情况就像下面这个样子：

标准的"X"和"O"，字母位于图像的正中央，并且比例合适，无变形

对于计算机来说，只要图像稍稍有一点变化，不是标准的，那么要解决这个问题还是不是那么容易的：

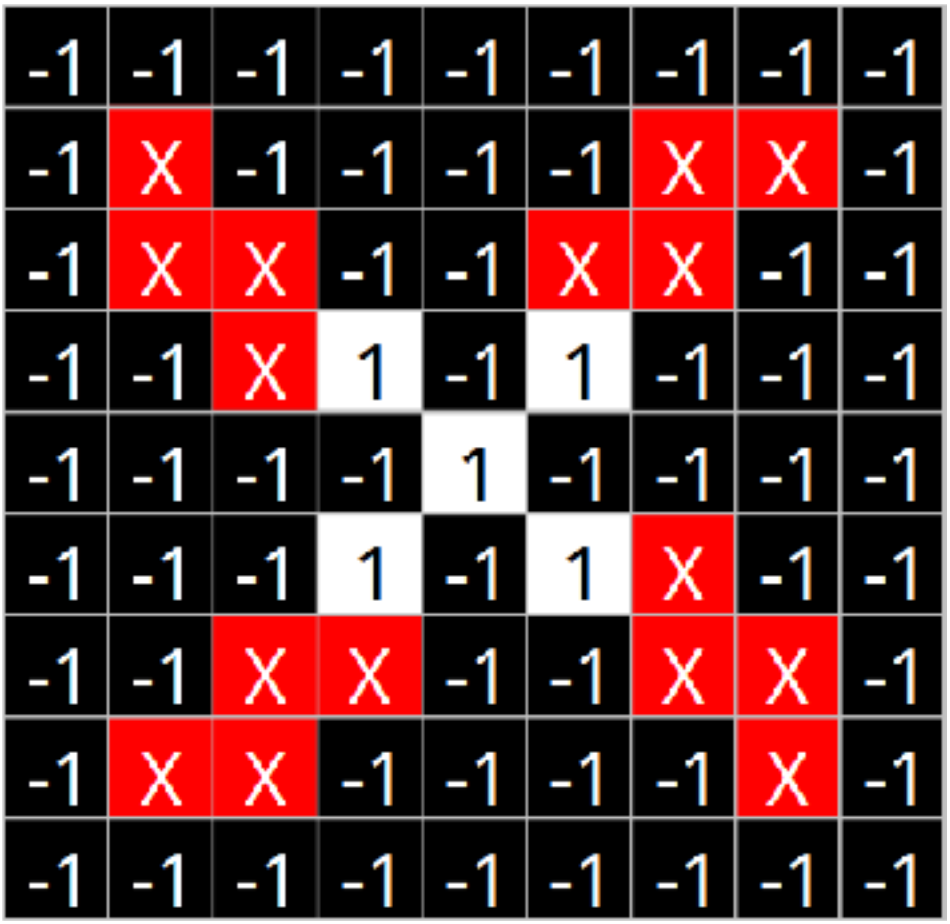


计算机要解决上面这个问题，一个比较天真的做法就是先保存一张"X"和"O"的标准图像（就像前面给出的例子），然后将其他的新给出的图像来和这两张标准图像进行对比，看看到底和哪一张图更匹配，就判断为哪个字母。但是这么做的话，其实是非常不可靠的，因为计算机还是比较死板的。在计算机的“视觉”中，一幅图看起来就像是一个二维的像素数组（可以想象成一个棋盘），每一个位置对应一个数字。在我们这个例子当中，像素值"1"代表白色，像素值"-1"代表黑色。

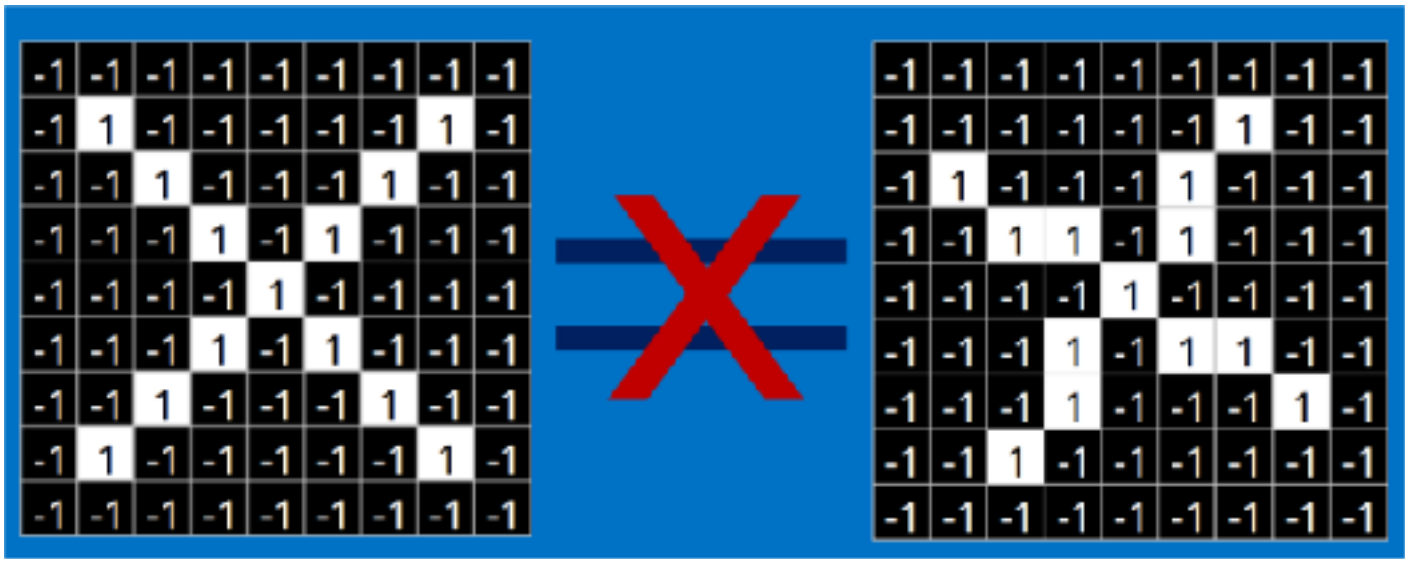


当比较两幅图的时候，如果有任何一个像素值不匹配，那么这两幅图就不匹配，至少对于计算机来说是这样的。

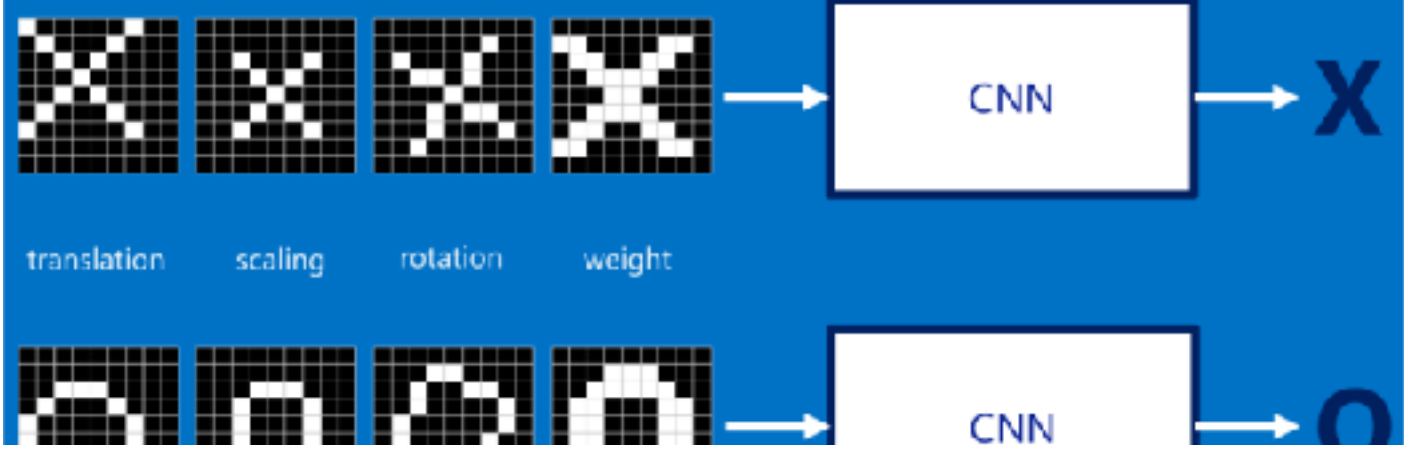
对于这个例子，计算机认为上述两幅图中的白色像素除了中间的3*3的小方格里面是相同的，其他四个角上都不同：



因此，从表面上看，计算机判别右边那幅图不是"X"，两幅图不同，得出结论：

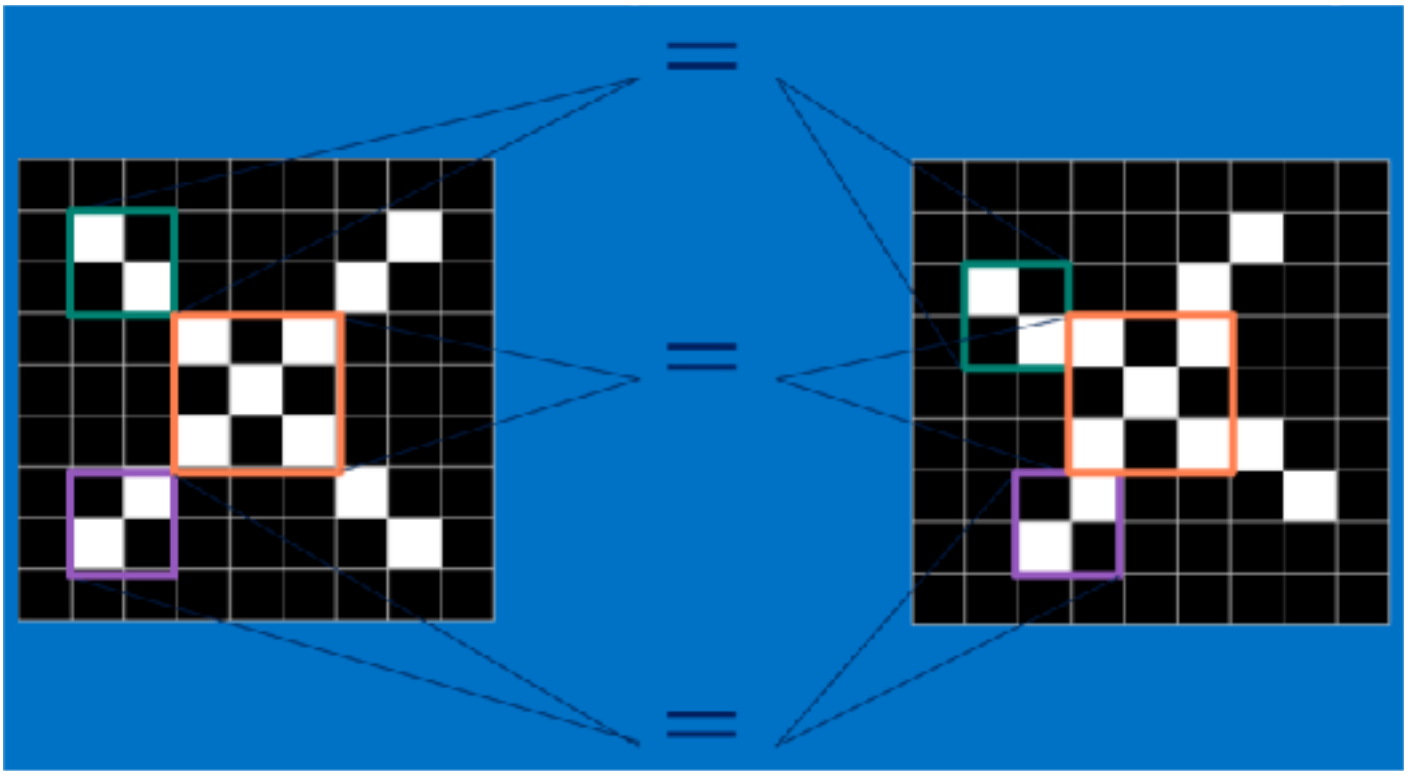


但是这么做，显得太不合理了。理想的情况下，我们希望，对于那些仅仅只是做了一些像平移，缩放，旋转，微变形等简单变换的图像，计算机仍然能够识别出图中的"X"和"O"。就像下面这些情况，我们希望计算机依然能够很快并且很准的识别出来：



这也就是CNN出现所要解决的问题。

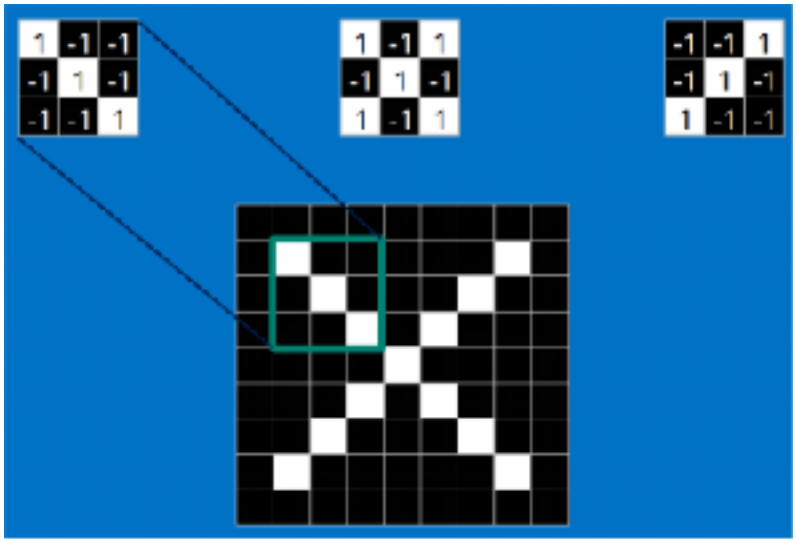
Features

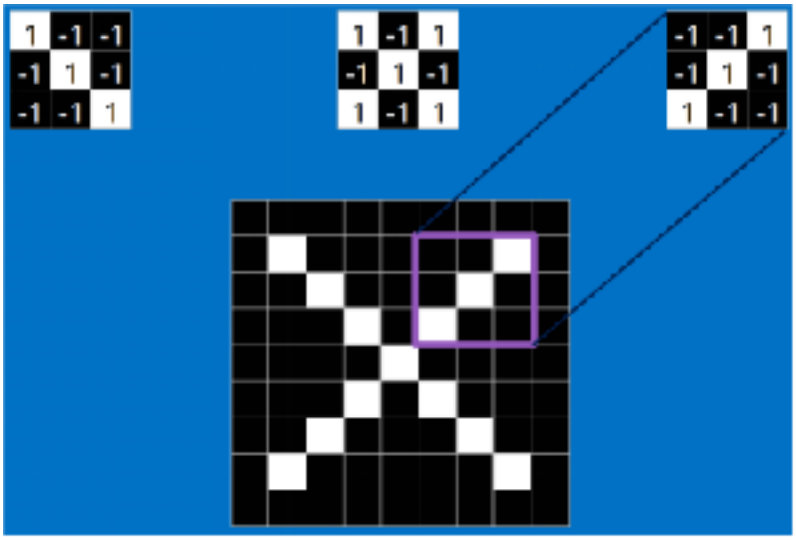


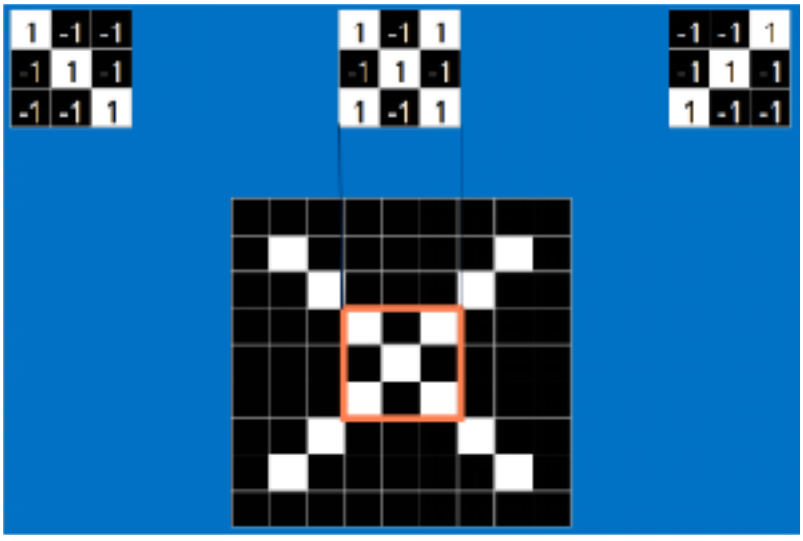
对于CNN来说，它是一块一块地来进行比对。它拿来比对的这个“小块”我们称之为Features（特征）。在两幅图中大致相同的位置找到一些粗糙的特征进行匹配，CNN能够更好的看到两幅图的相似性，相比起传统的整幅图逐一比对的方法。

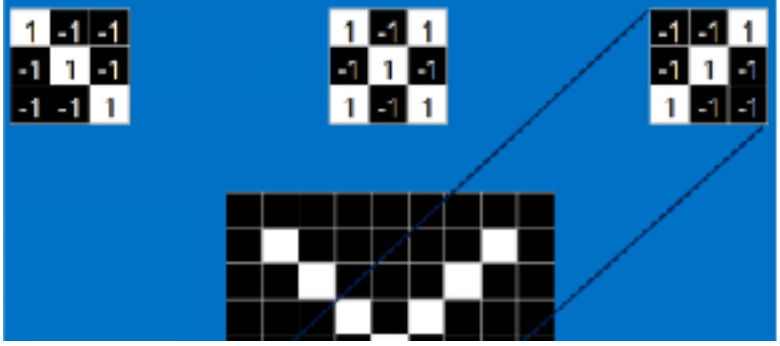
每一个feature就像是一个小图（就是一个比较小的有值的二维数组）。不同的Feature匹配图像中不同的特征。在字母"X"的例子中，那些由对角线和交叉线组成的features基本上能够识别出大多数"X"所具有的重要特征。

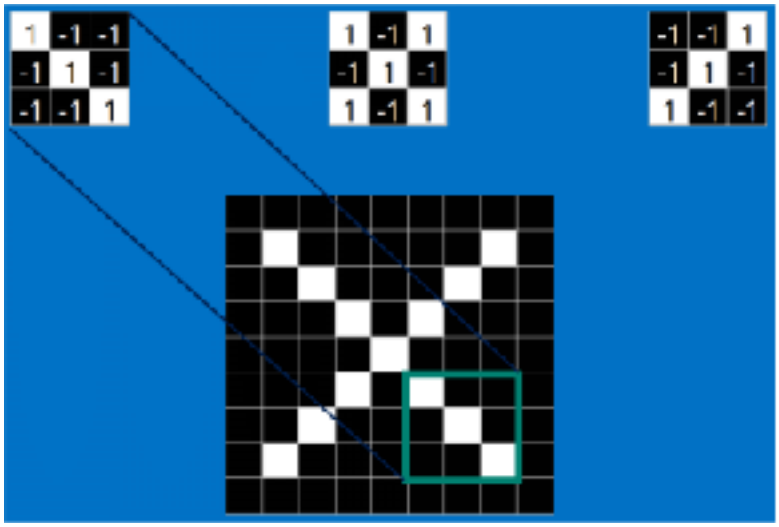
这些features很有可能就是匹配任何含有字母"X"的图中字母X的四个角和它的中心。那么具体到底是怎么匹配的呢？如下：



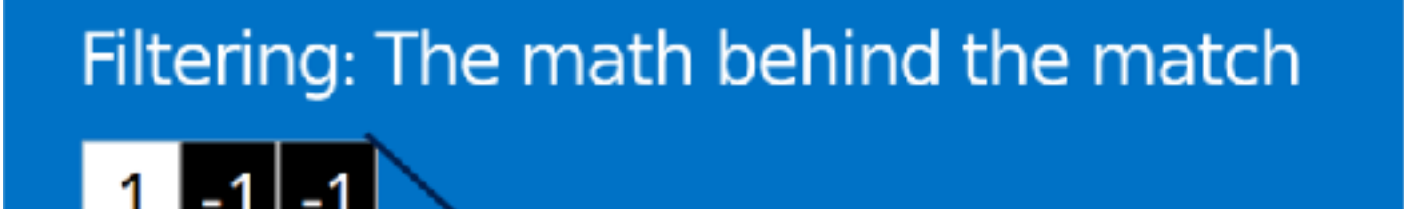






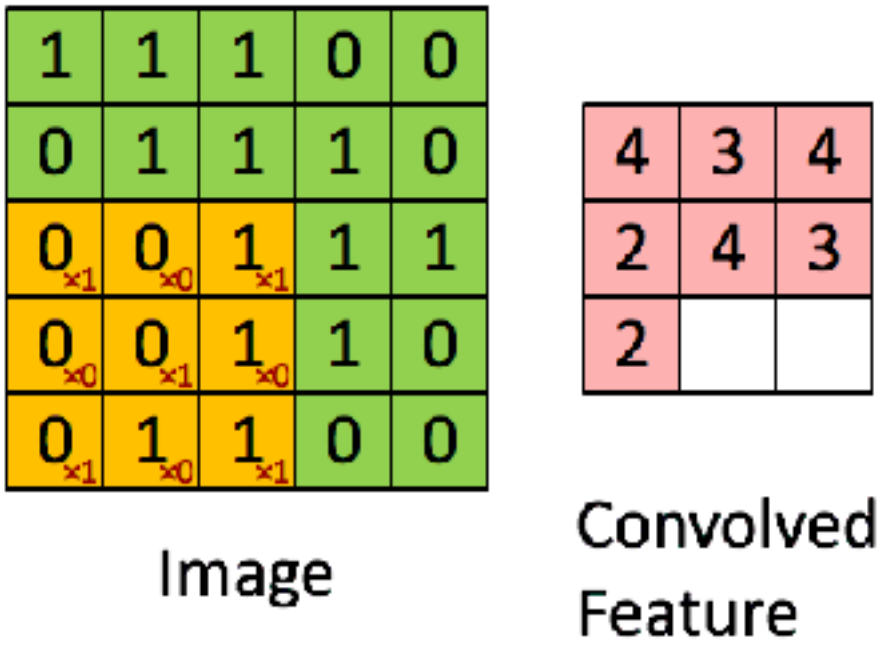


看到这里是不是有了一点头目呢。但其实这只是第一步，你知道了这些Features是怎么在原图上面进行匹配的。但是你还不知道在这里面究竟进行的是怎样的数学计算，比如这个下面3*3的小块到底干了什么?



接下来就跟进介绍里面的数学操作，也就是我们常说的“卷积”操作。

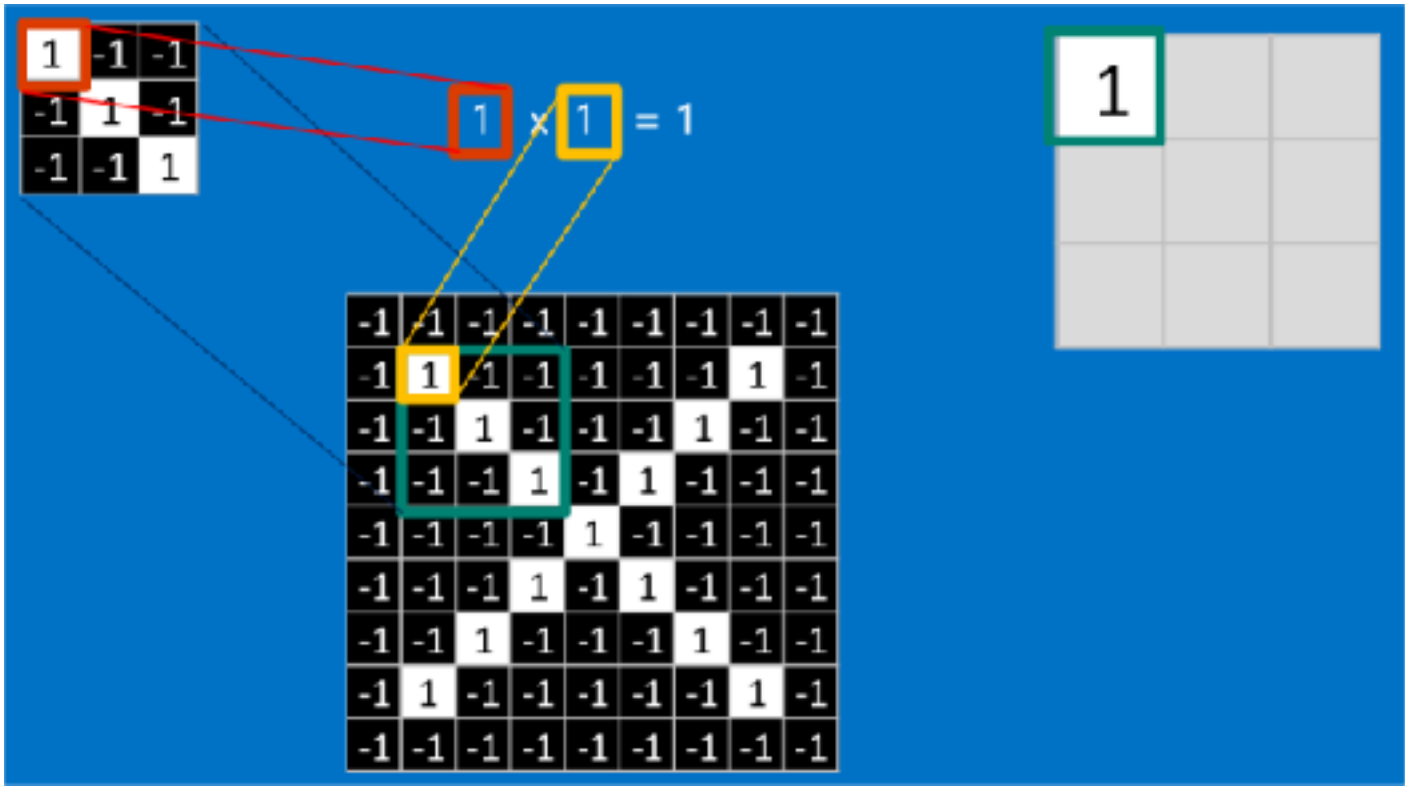
卷积(Convolution)

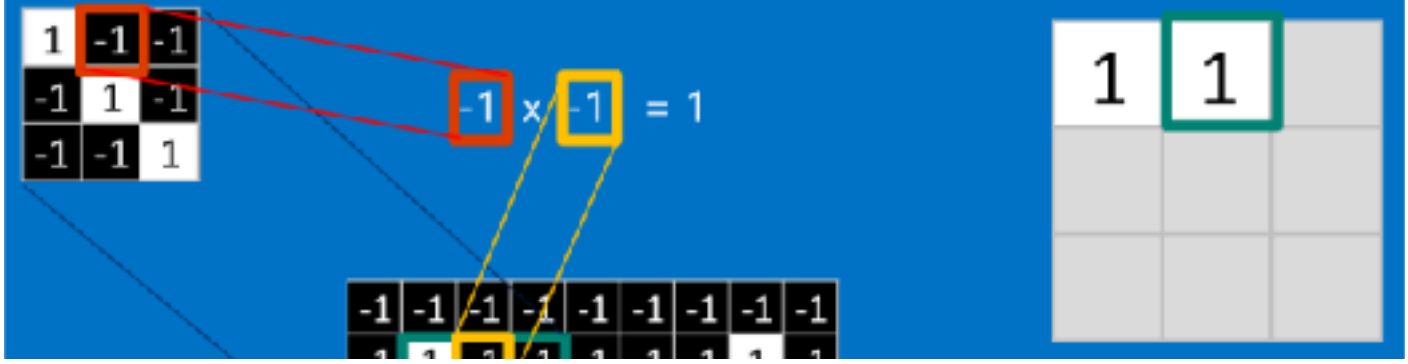


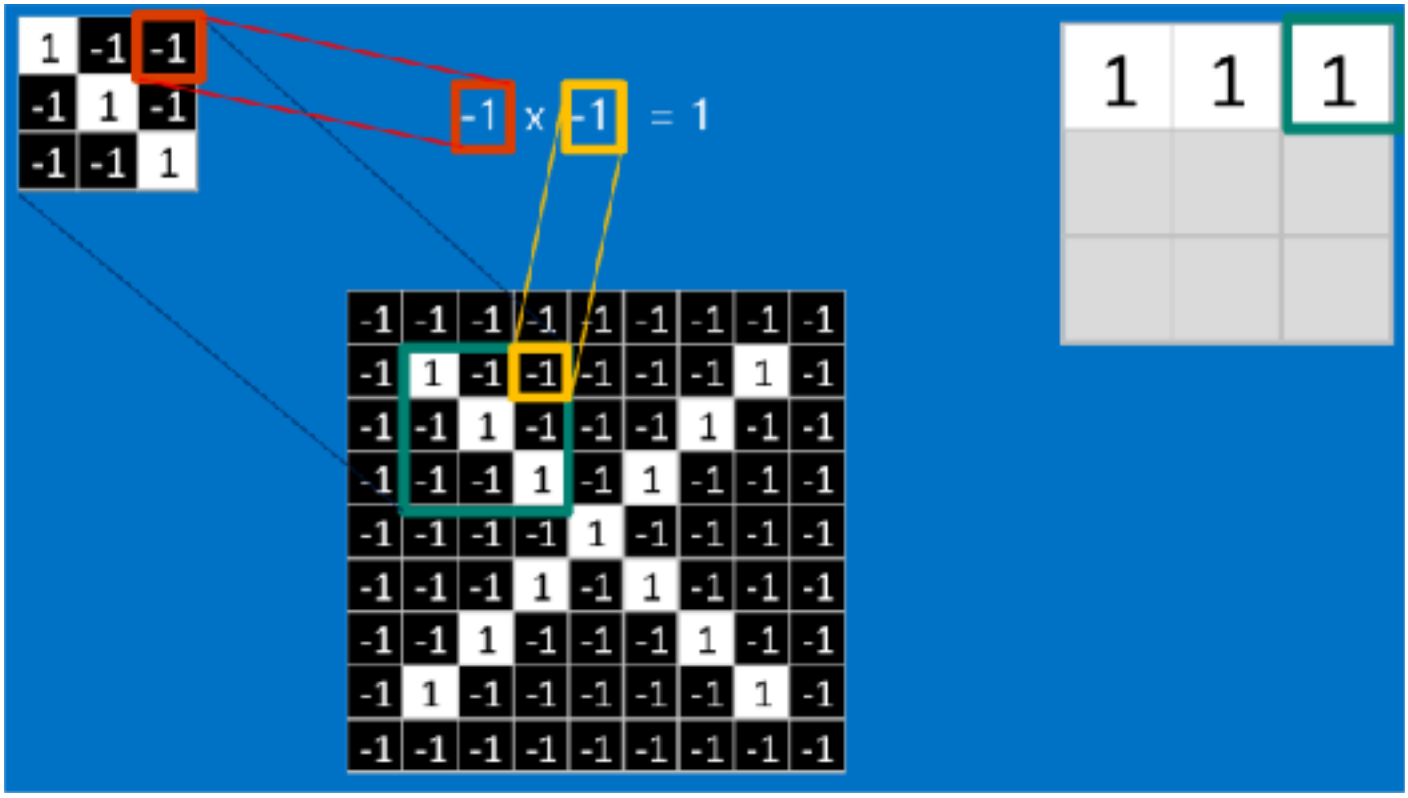
Convolution

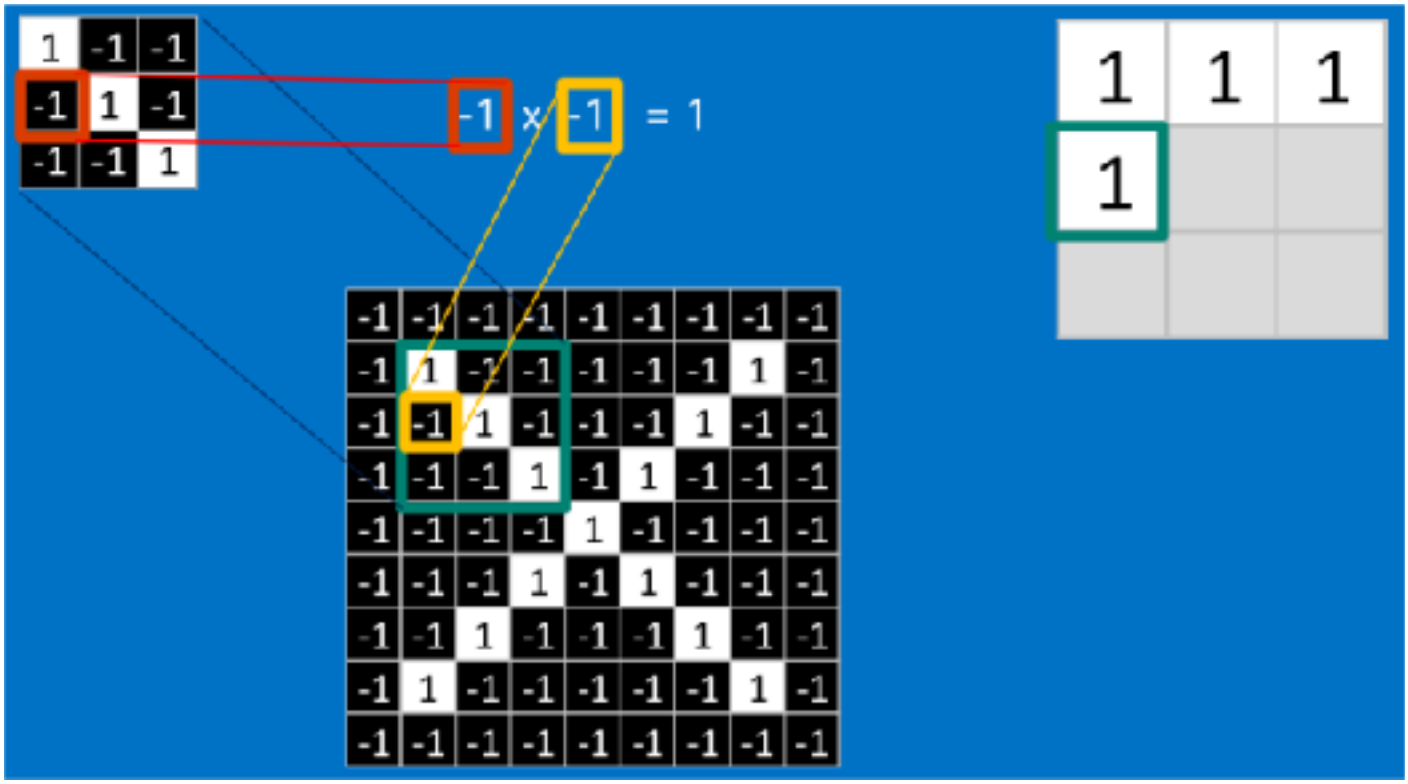
当给你一张新的图时，CNN并不能准确地知道这些features到底要匹配原图的哪些部分，所以它会在原图中每一个可能的位置进行尝试。这样在原始整幅图上每一个位置进行匹配计算，我们相当于把这个feature变成了一个过滤器。这个我们用来匹配的过程就被称为卷积操作，这也就是卷积神经网络名字的由来。

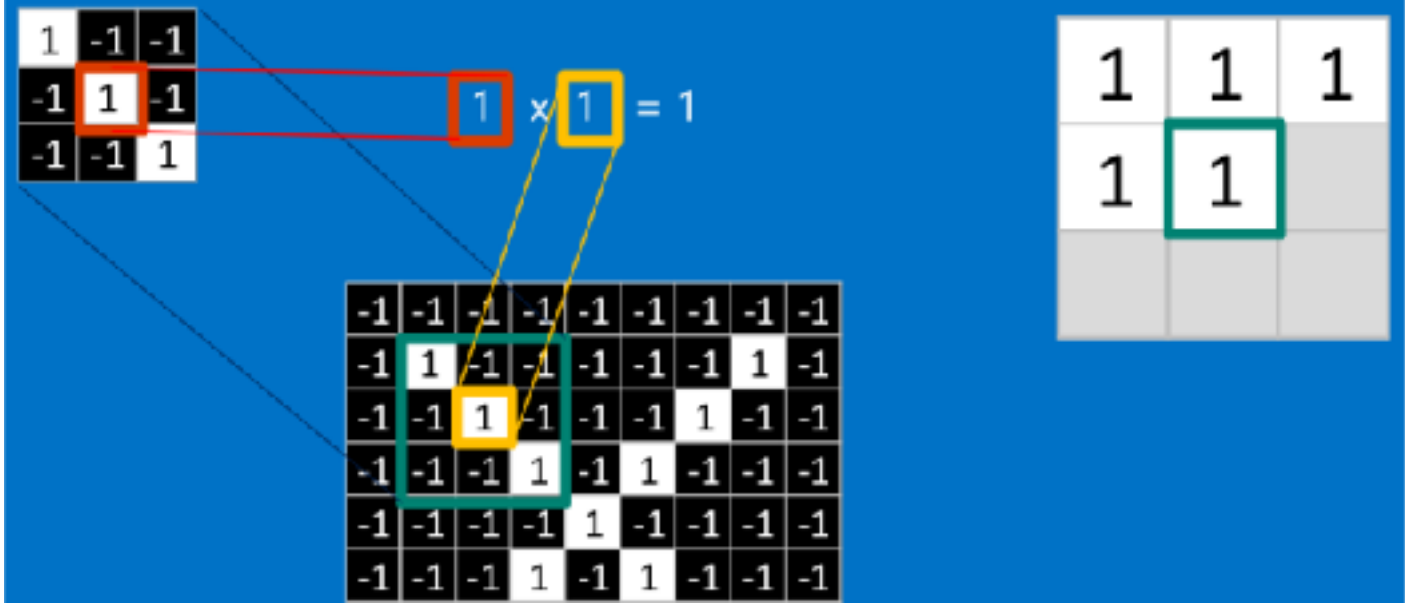
这个卷积操作背后的数学知识其实非常的简单。要计算一个feature和其在原图上对应的某一小块的结果，只需要简单地将两个小块内对应位置的像素值进行乘法运算，然后将整个小块内乘法运算的结果累加起来，最后再除以小块内像素点总个数即可。如果两个像素点都是白色（也就是值均为1），那么1*1 = 1，如果均为黑色，那么(-1)*(-1) = 1。不管哪种情况，每一对能够匹配上的像素，其相乘结果为1。类似地，任何不匹配的像素相乘结果为-1。如果一个feature（比如n*n）内部所有的像素都和原图中对应一小块（n*n）匹配上了，那么它们对应像素值相乘再累加就等于n^2，然后除以像素点总个数n^2，结果就是1。同理，如果每一个像素都不匹配，那么结果就是-1。具体过程如下：

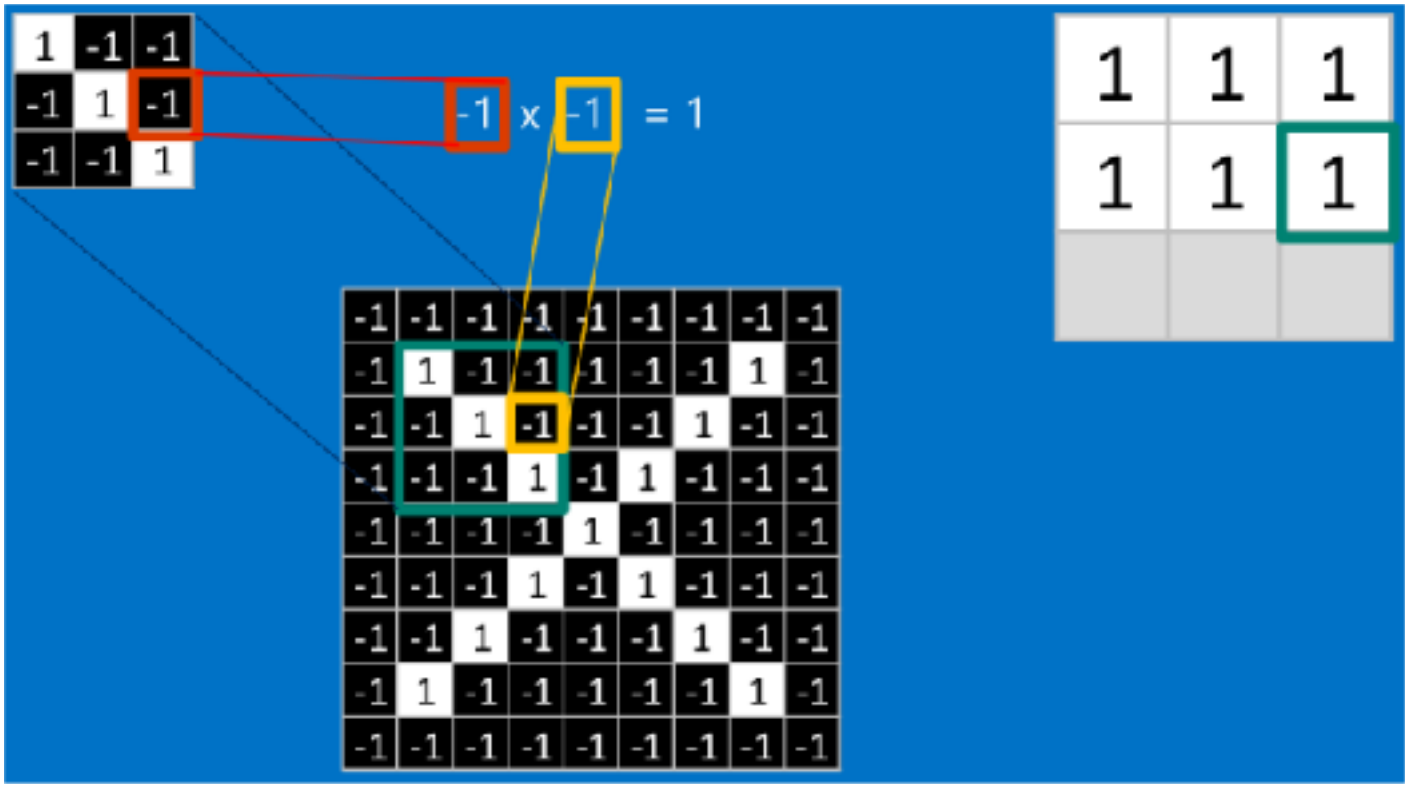


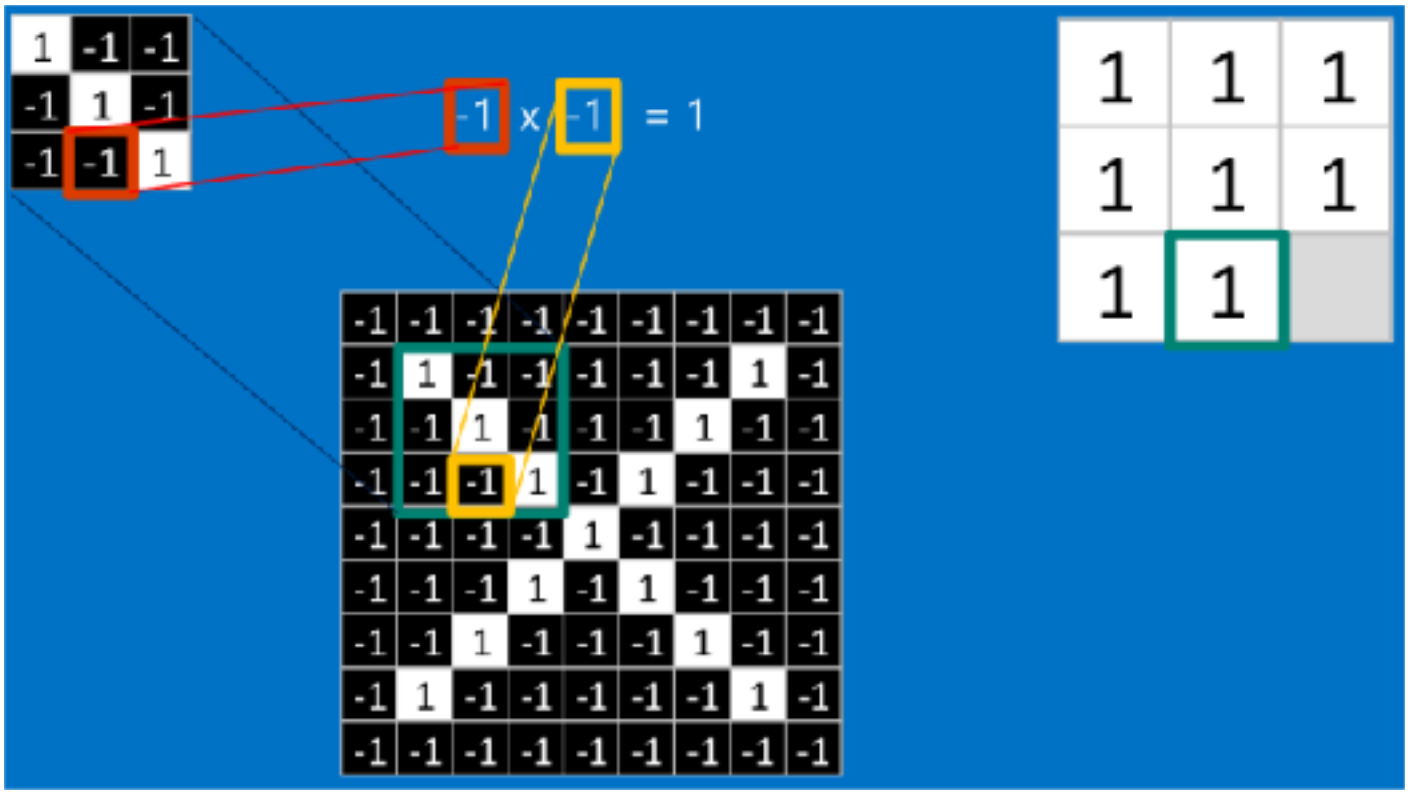

1 -1 -1
-1 1 -1
-1 -1 1
-1 x -1 = 1
1 1

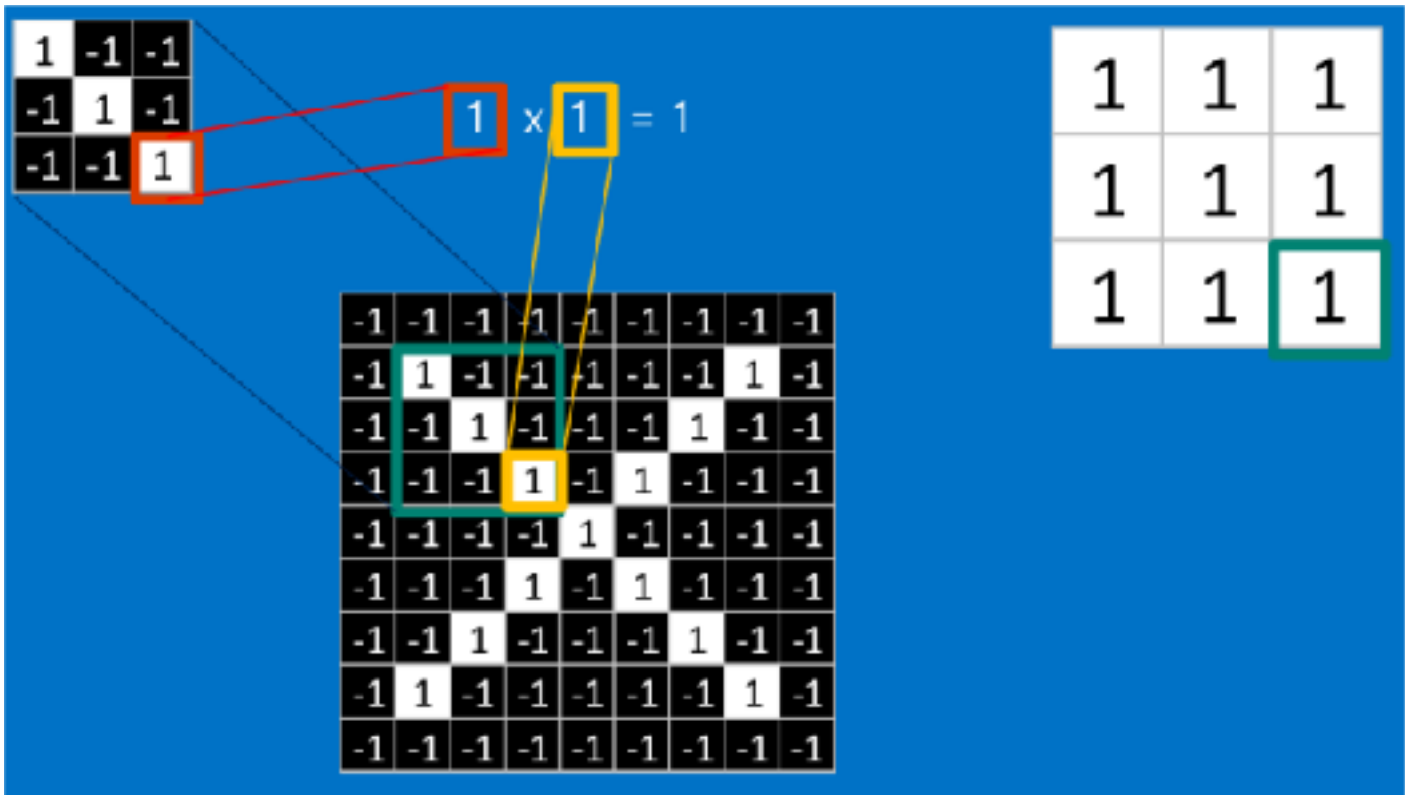

1 -1 -1
-1 1 -1
-1 -1 1
-1 x -1 = 1
1 1 1


1 -1 -1
-1 1 -1
-1 -1 1
-1 x -1 = 1
1 1 1
1

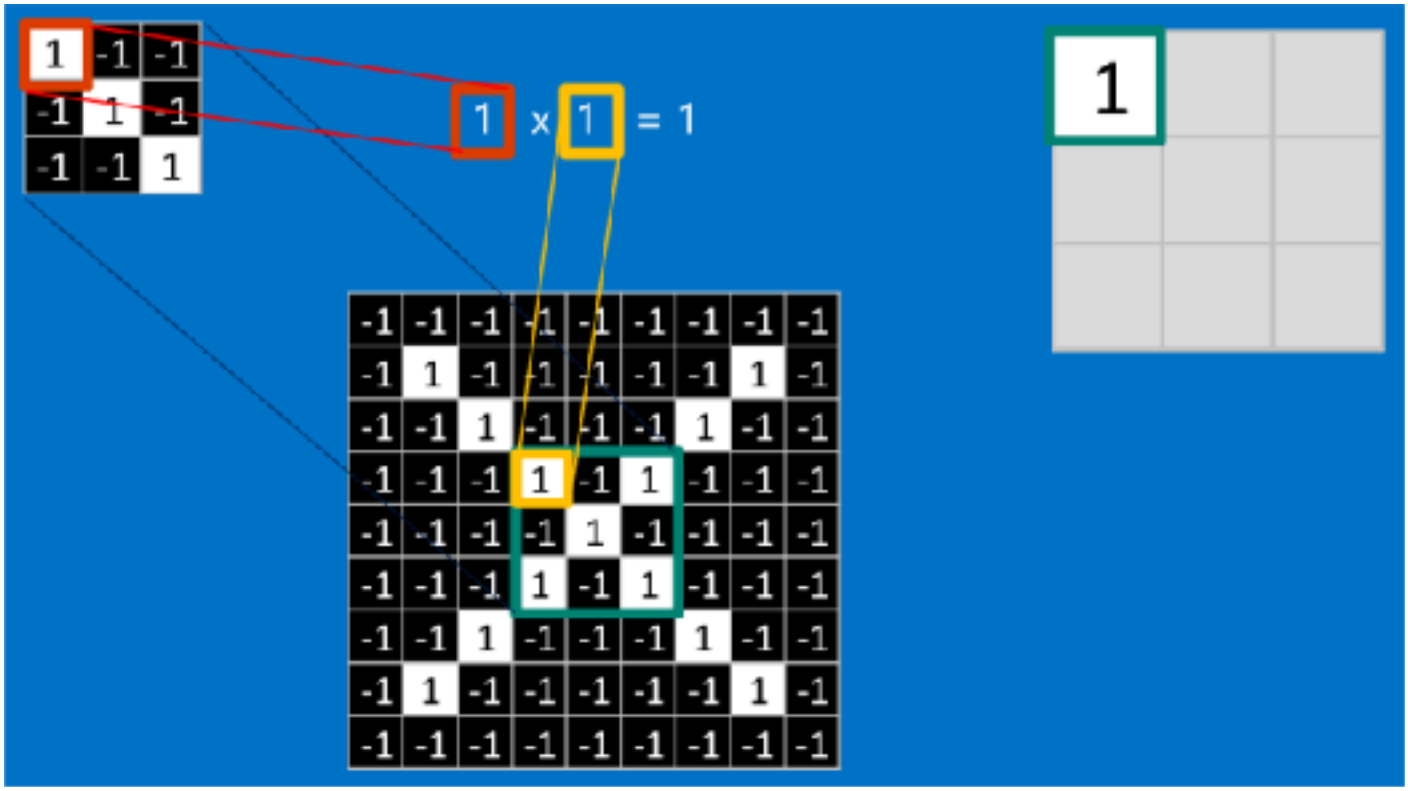

1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
1 1 1
1 1

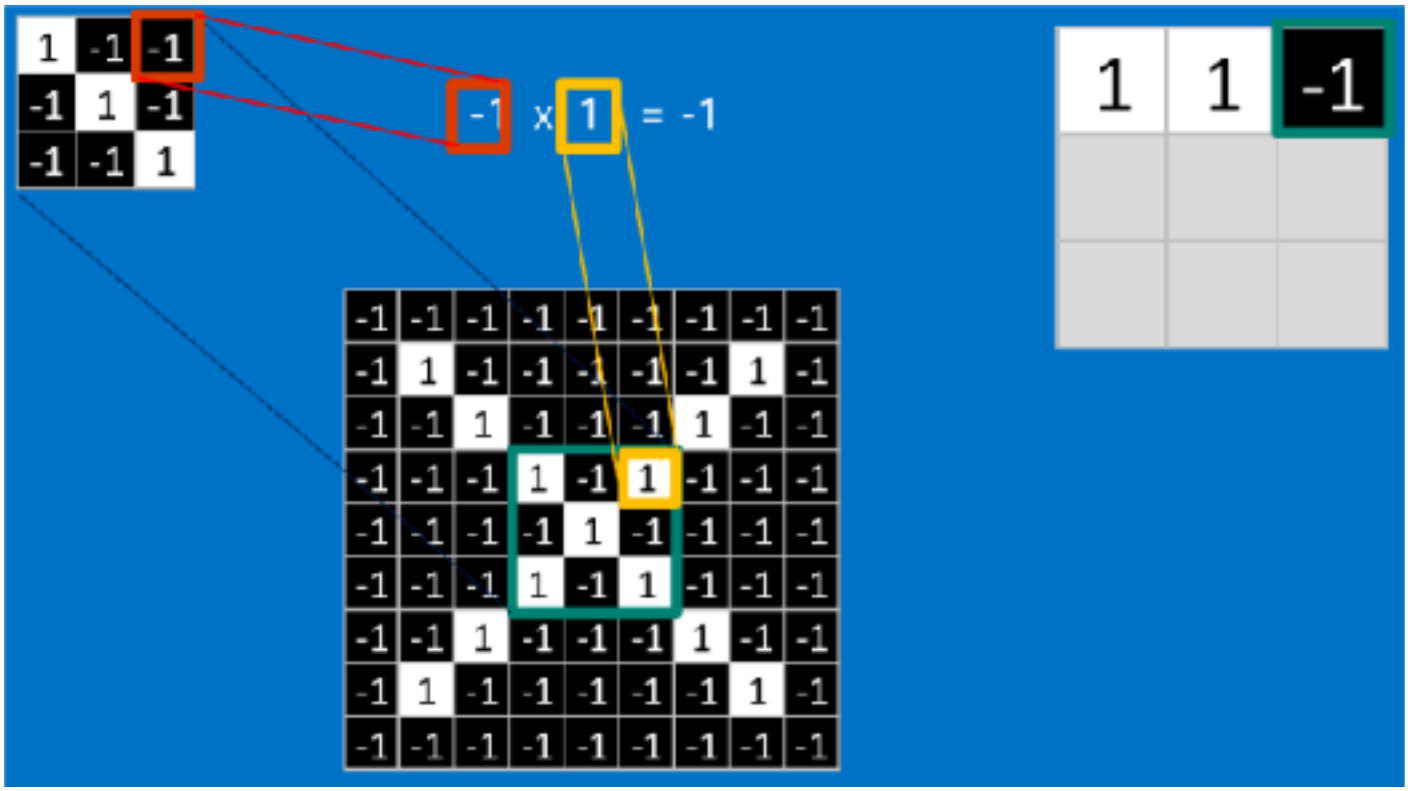

1 -1 -1
-1 1 -1
-1 -1 1
-1 x -1 = 1
1 1 1
1 1 1

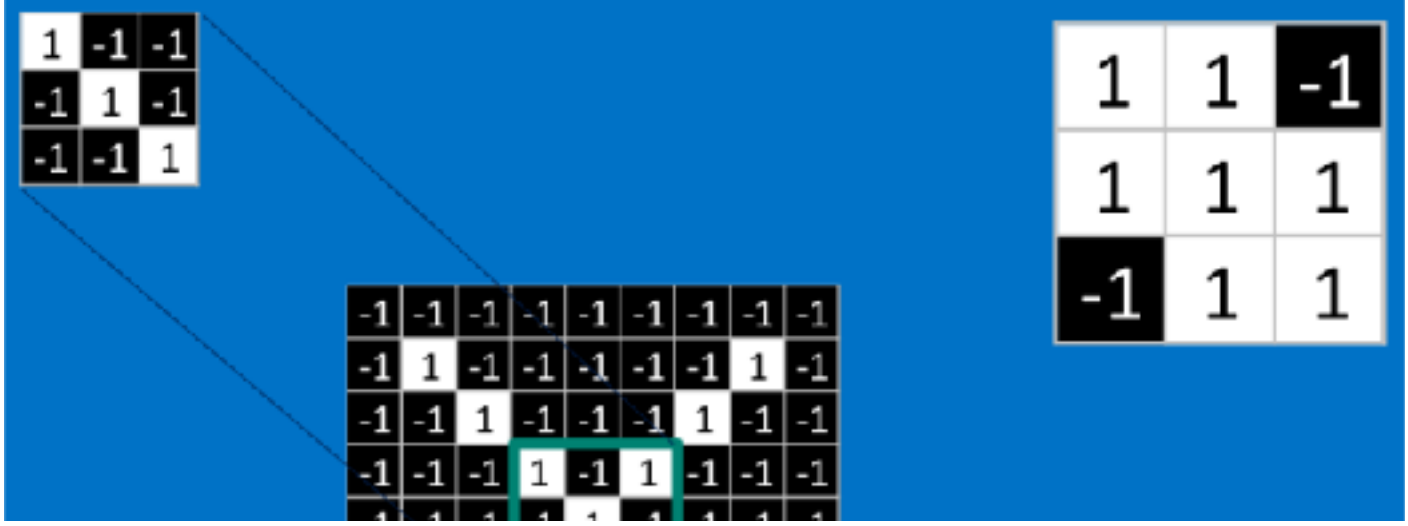

1 -1 -1
-1 1 -1
-1 -1 1
-1 x -1 = 1
1 1 1
1 1 1
1 1

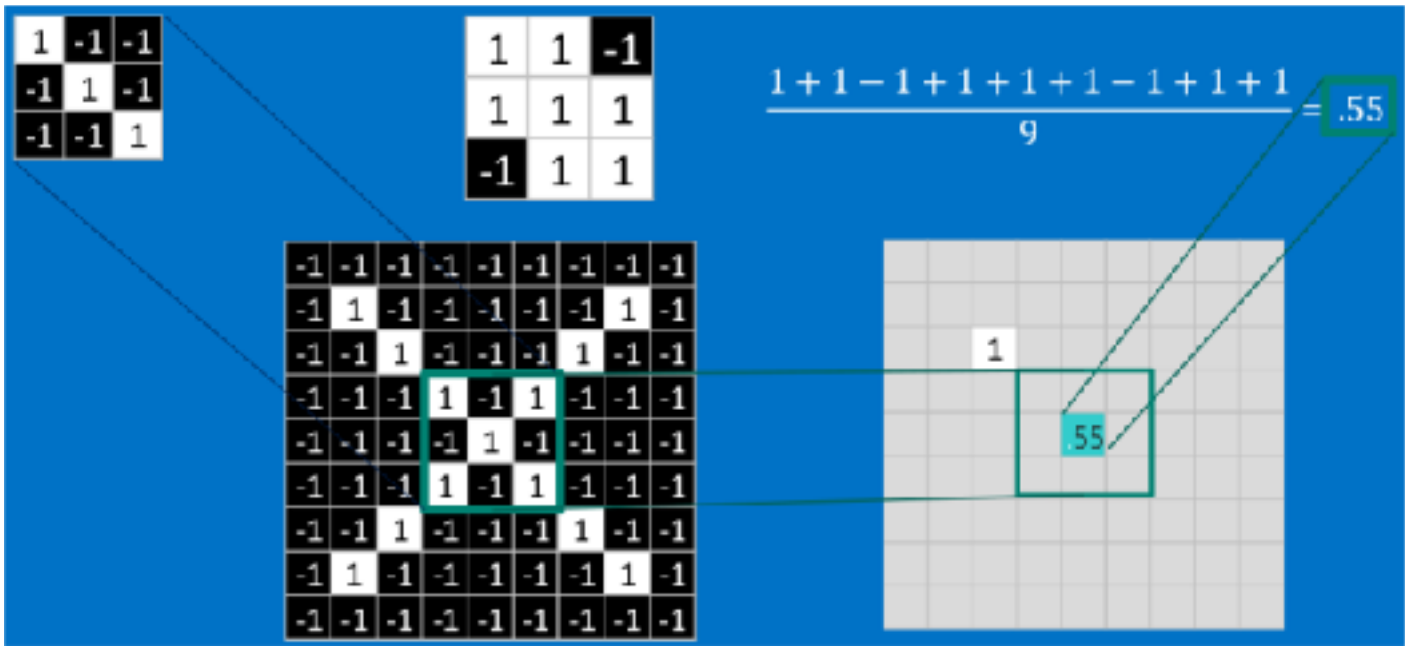

1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
1 1 1
1 1 1
1 1 1

对于中间部分，也是一样的操作：

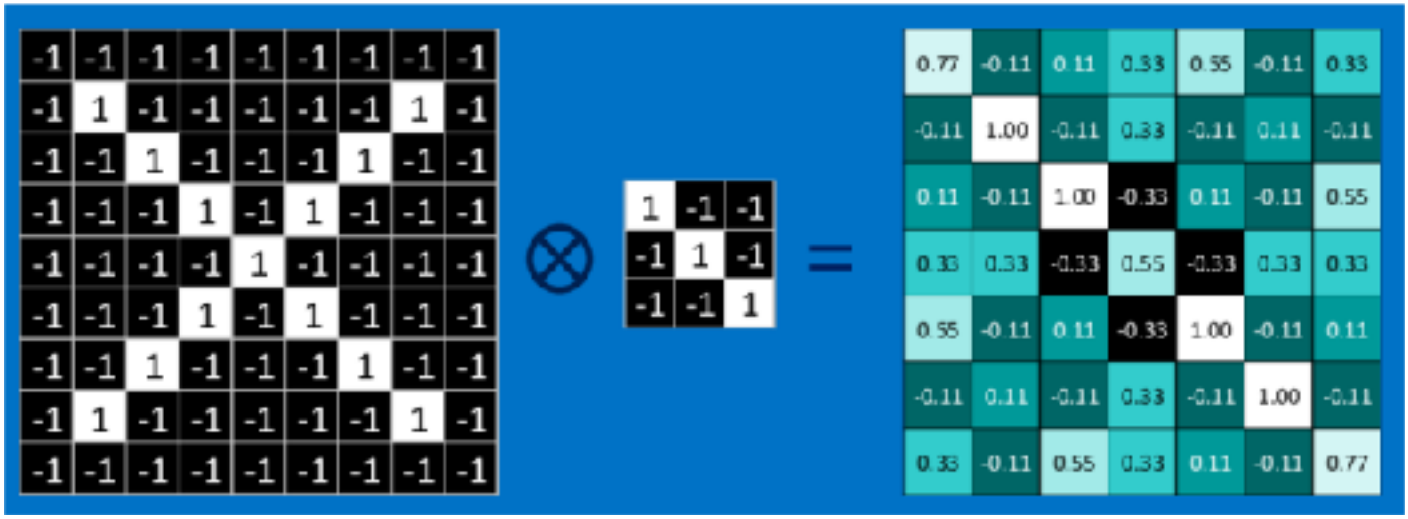






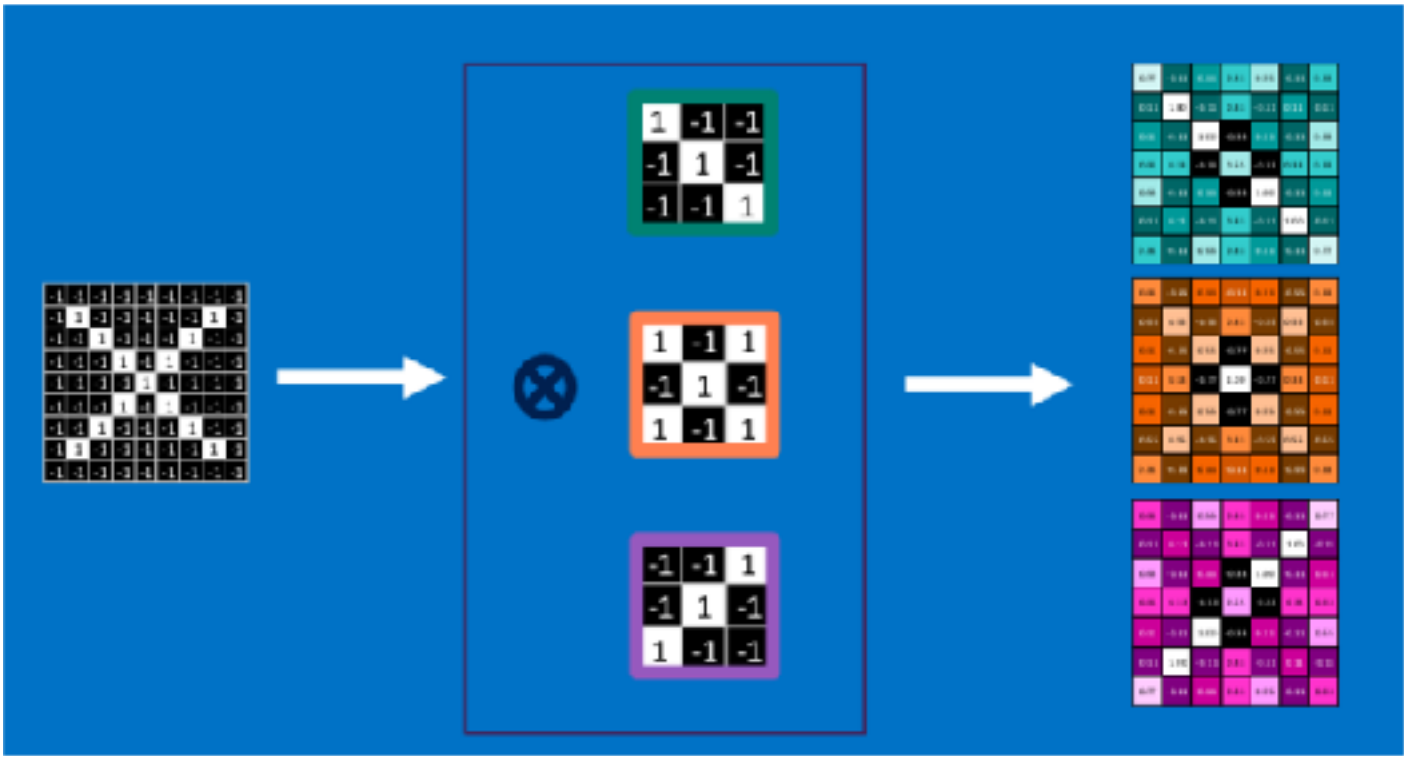


最后整张图算完，大概就像下面这个样子：

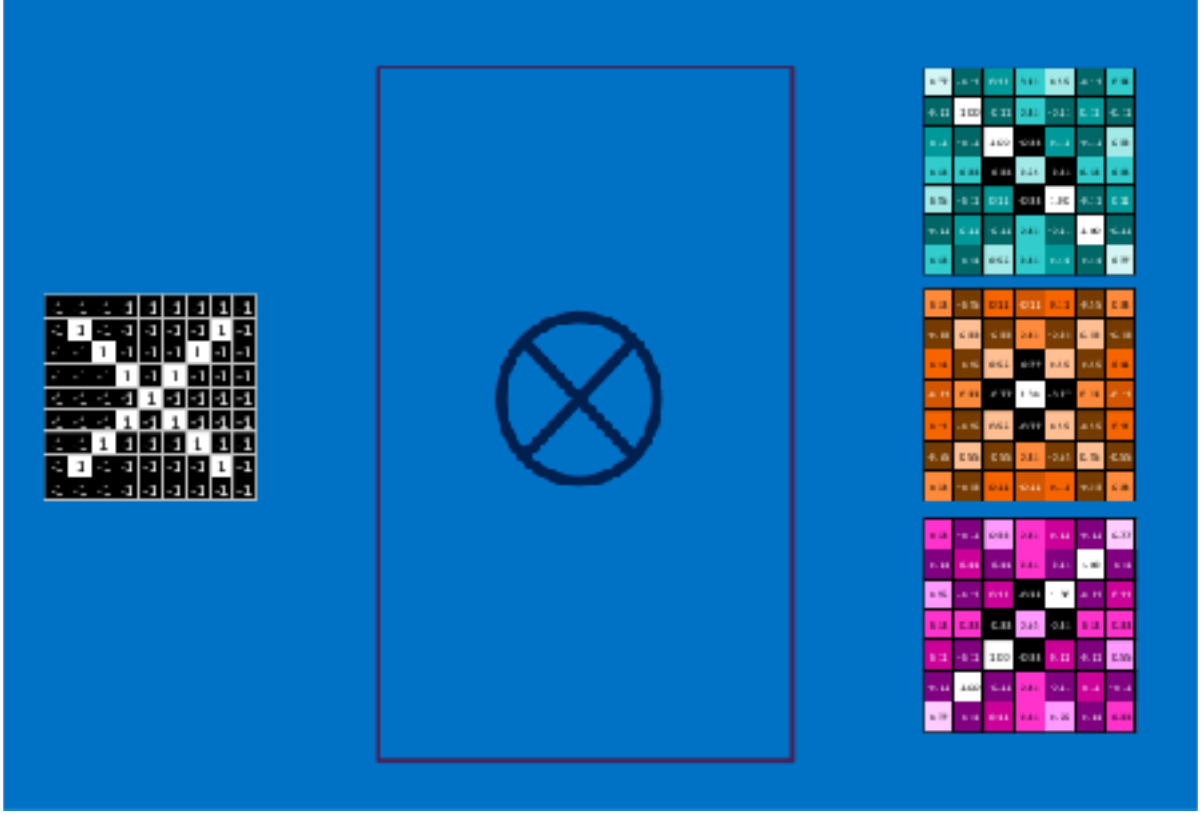


然后换用其他feature进行同样的操作，最后得到的结果就是这样了：

为了完成我们的卷积，我们不断地重复着上述过程，将feature和图中每一块进行卷积操作。最后通过每一个feature的卷积操作，我们会得到一个新的二维数组。这也可以理解为对原始图像进行过滤的结果，我们称之为feature map，它是每一个feature从原始图像中提取出来的“特征”。其中的值，越接近为1表示对应位置和feature的匹配越完整，越是接近-1，表示对应位置和feature的反面匹配越完整，而值接近0的表示对应位置没有任何匹配或者说没有什么关联。

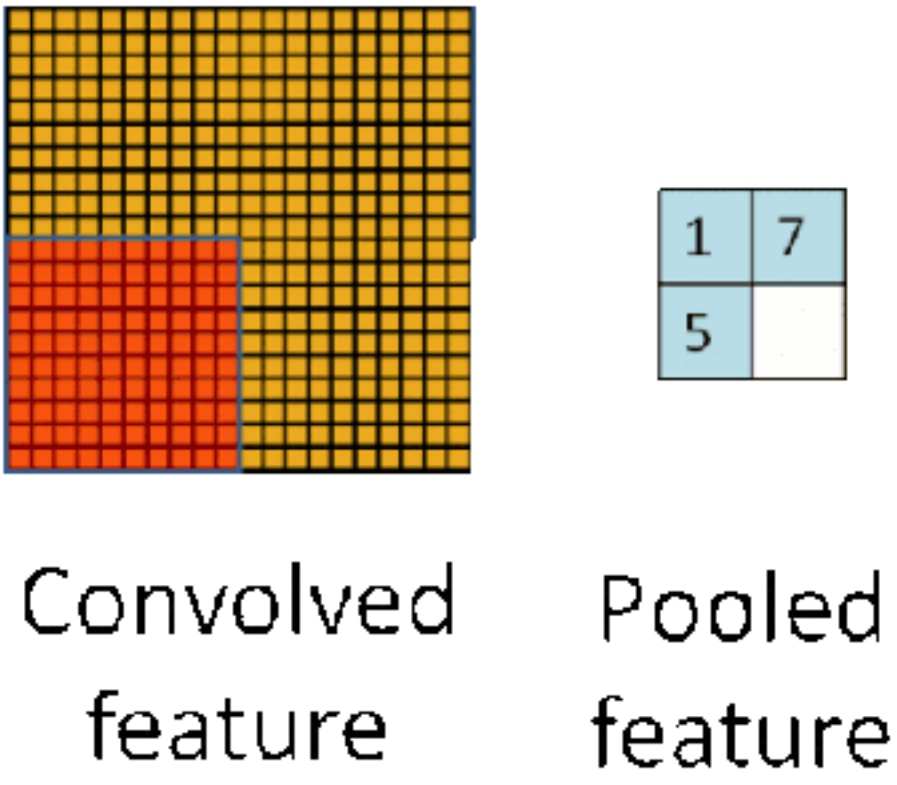


这样我们的原始图，经过不同feature的卷积操作就变成了一系列的feature map。我们可以很方便，直观地将这整个操作视为一个单独的处理过程。在CNN中，我们称之为卷积层(convolution layer)，这样你可能很快就会想到后面肯定还有其他的layer。没错，后面会提到。我们可以将卷积层看成下面这个样子：



因此可想而知，CNN其实做的操作也没什么复杂的。但是尽管我们能够以这一点篇幅就描述了CNN的工作，其内部的加法，乘法和除法操作的次数其实会增加地很快。从数学的角度来说，它们会随着图像的大小，每一个filter的大小和filter的数目呈线性增长。由于有这么多因素的影响，很容易使得这个问题的计算量变得相当的庞大，这也难怪很多微处理器制造商现在都在生产制造专业的芯片来跟上CNN计算的需求。

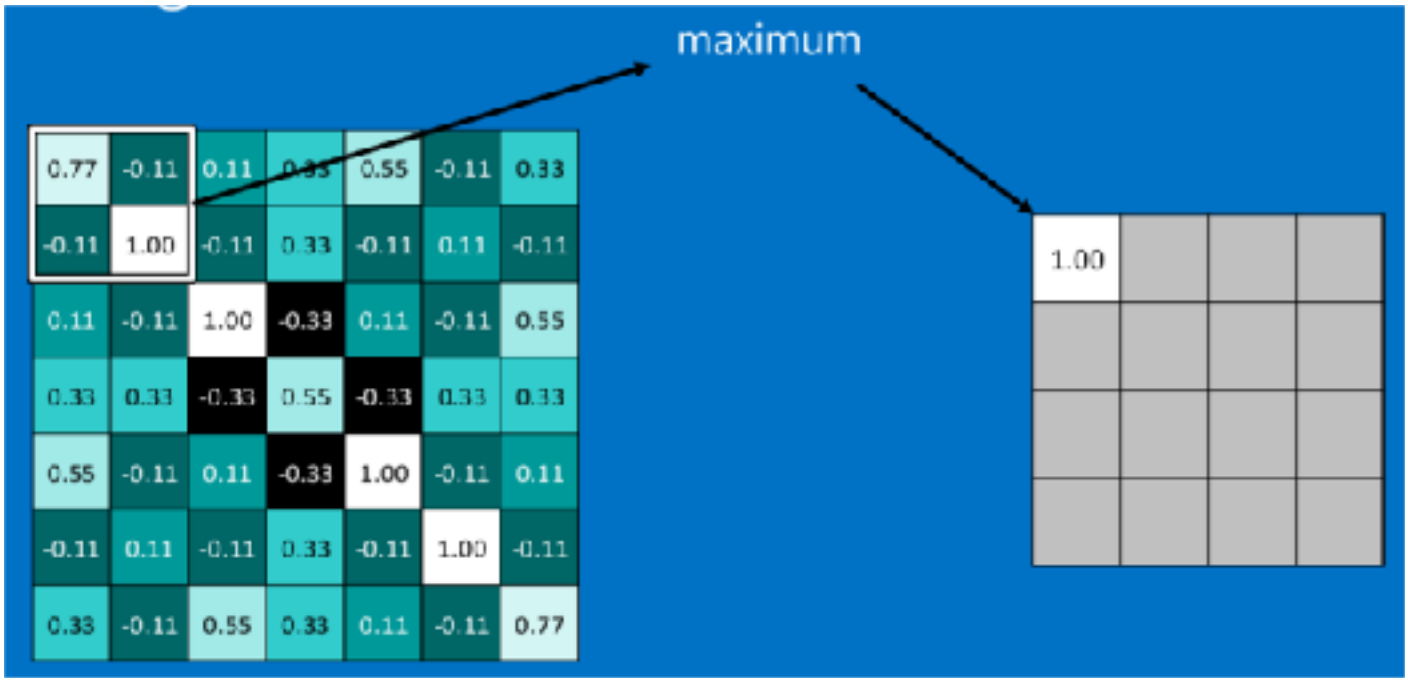
池化(Pooling)

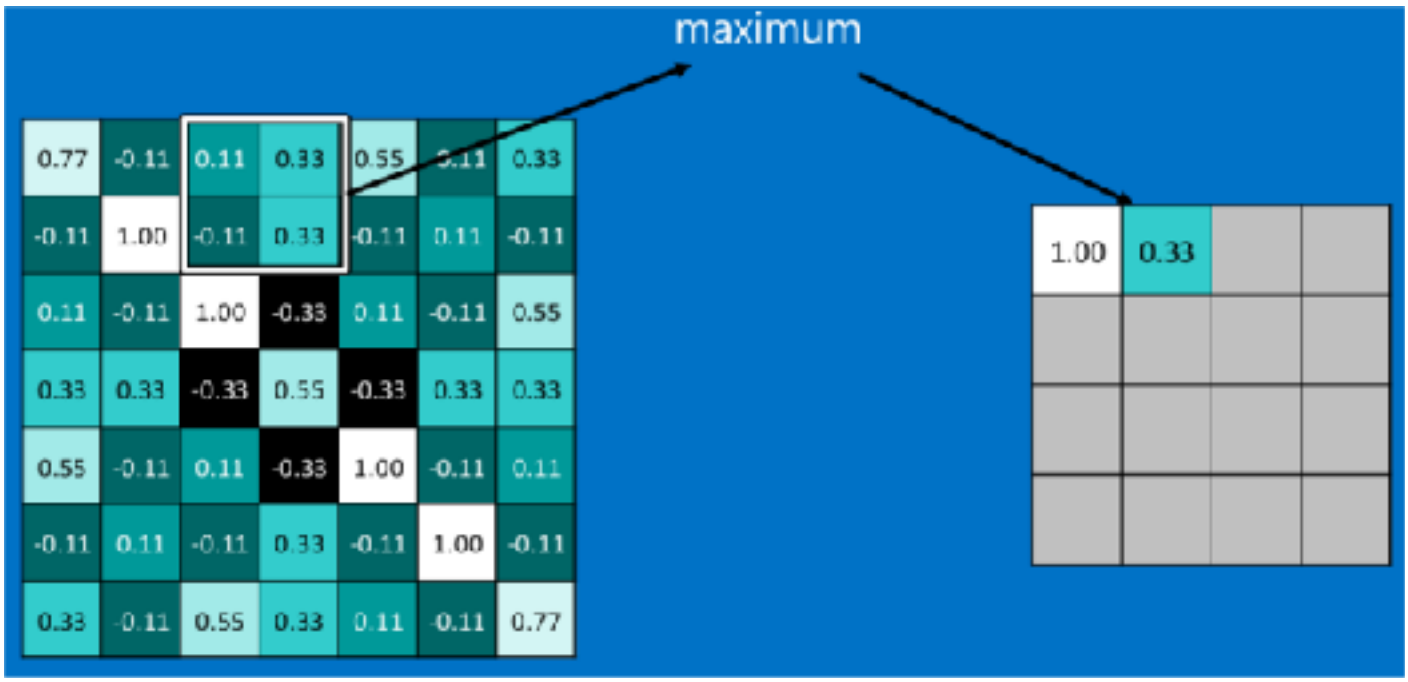


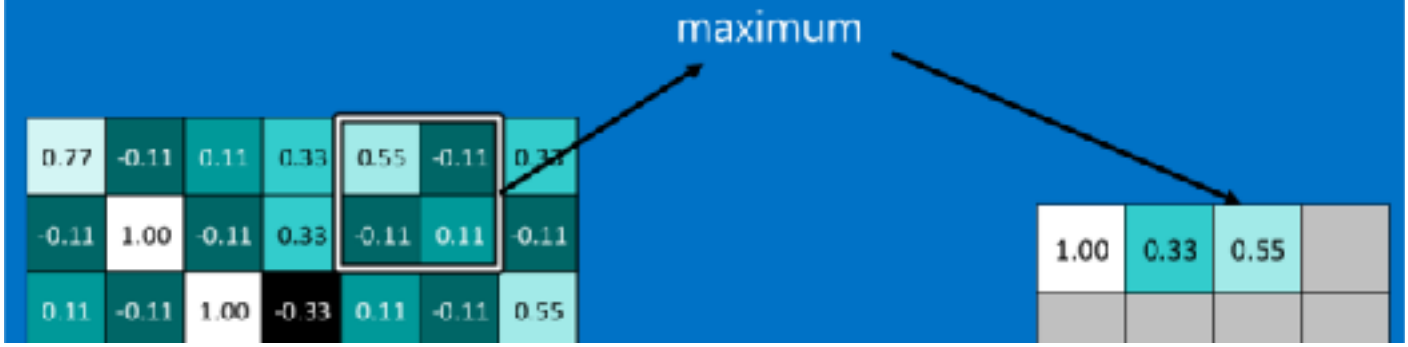
Pooling

CNN中使用的另一个有效的工具被称为“池化(Pooling)”。池化可以将一幅大的图像缩小，同时又保留其中的重要信息。池化背后的数学顶多也就是小学二年级水平。它就是将输入图像进行缩小，减少像素信息，只保留重要信息。通常情况下，池化都是2*2大小，比如对于max-pooling来说，就是取输入图像中2*2大小的块中的最大值，作为结果的像素值，相当于将原始图像缩小了4倍。(注：同理，对于average-pooling来说，就是取2*2大小块的平均值作为结果的像素值。)

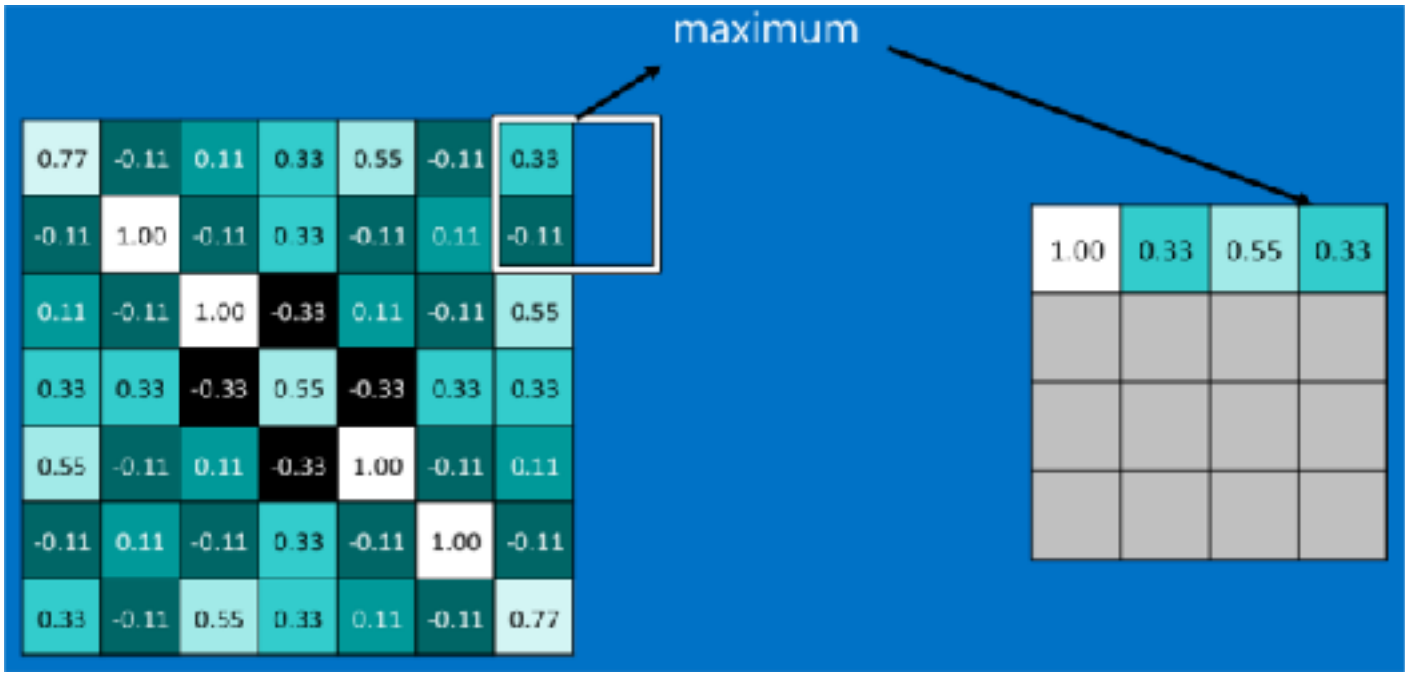
对于本文的这个例子，池化操作具体如下：

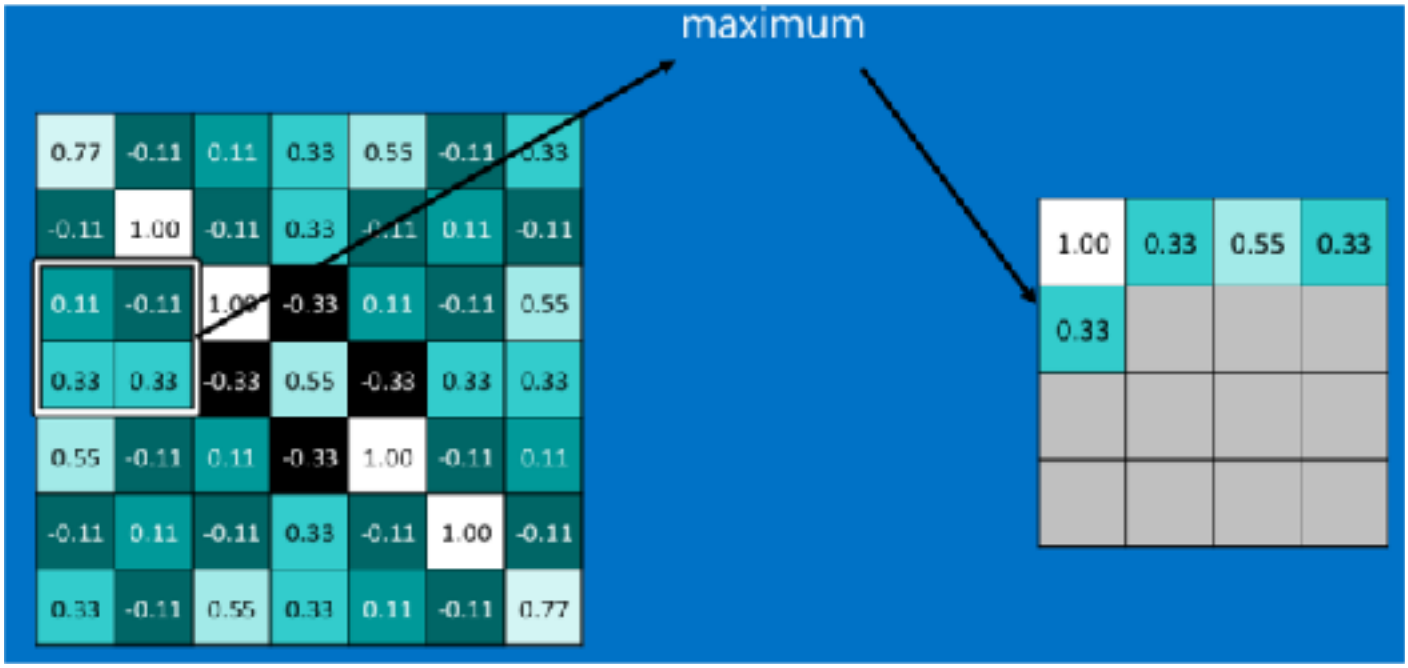




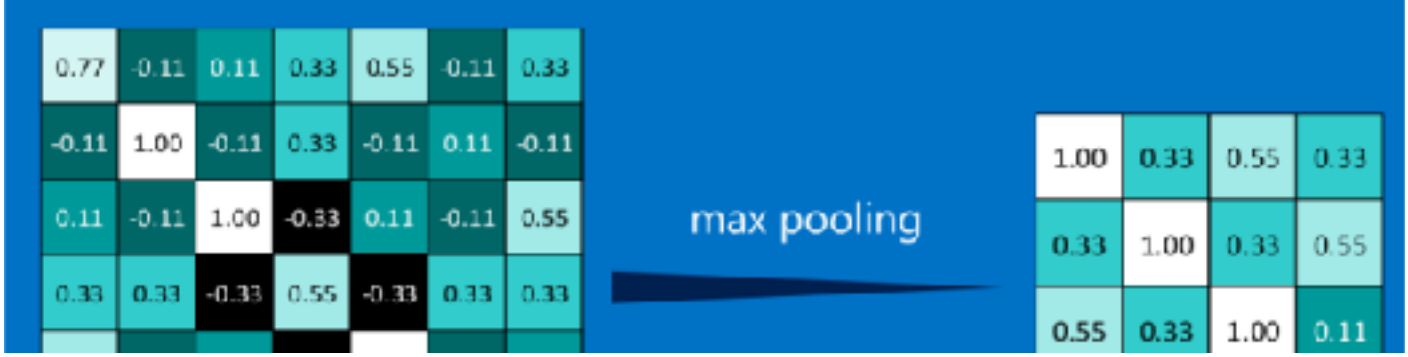


不足的外面补"0"：

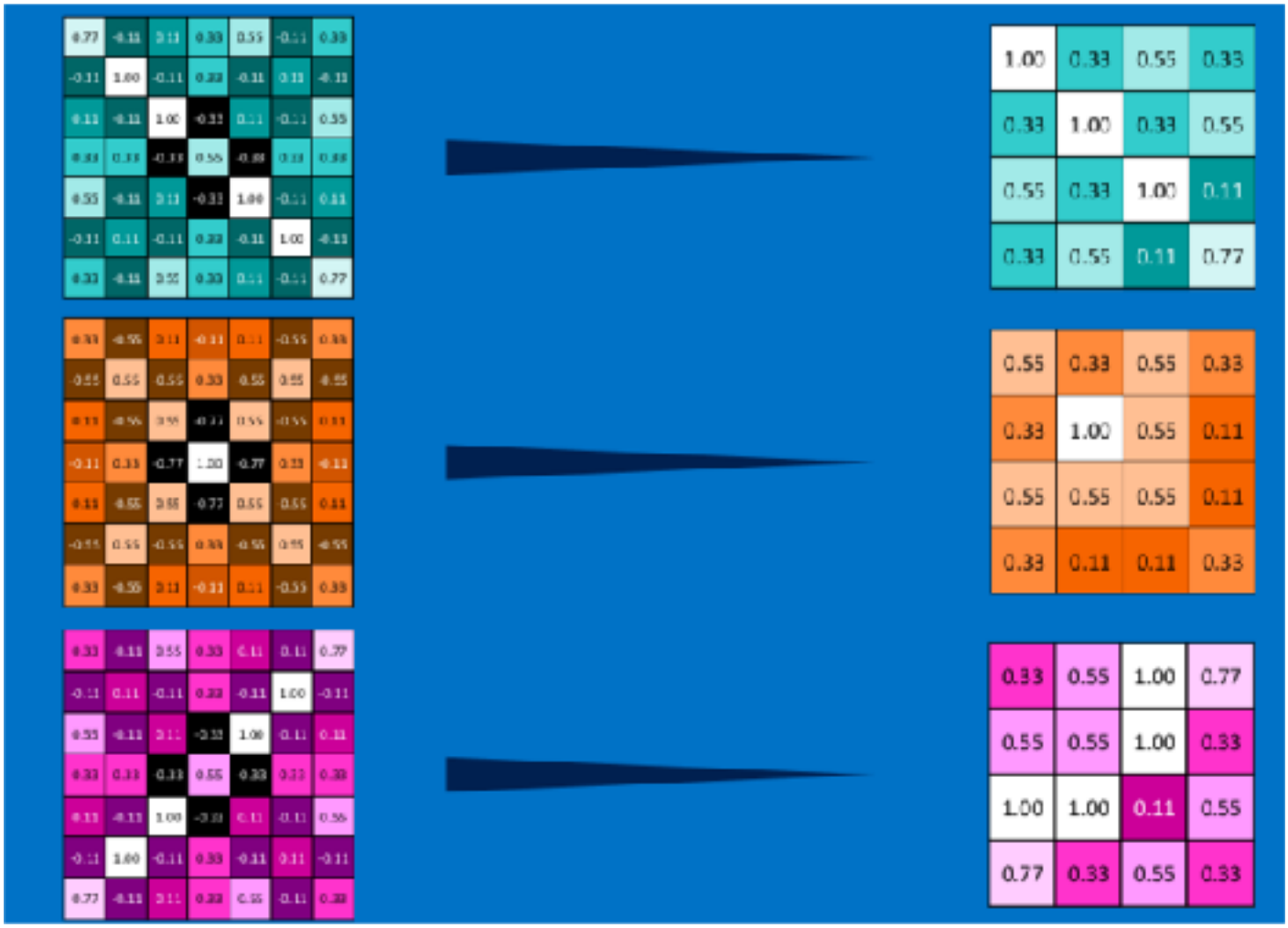




经过最大池化操作（比如2*2大小）之后，一幅图就缩小为原来的四分之一了：

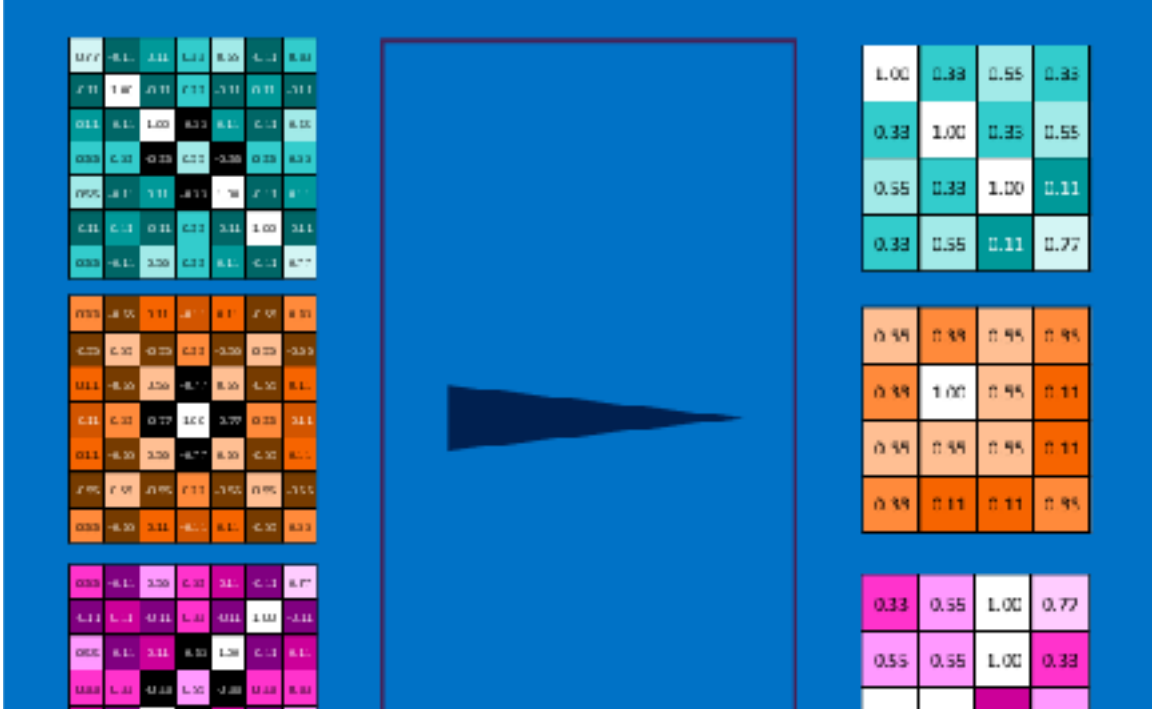


然后对所有的feature map执行同样的操作，得到如下结果：



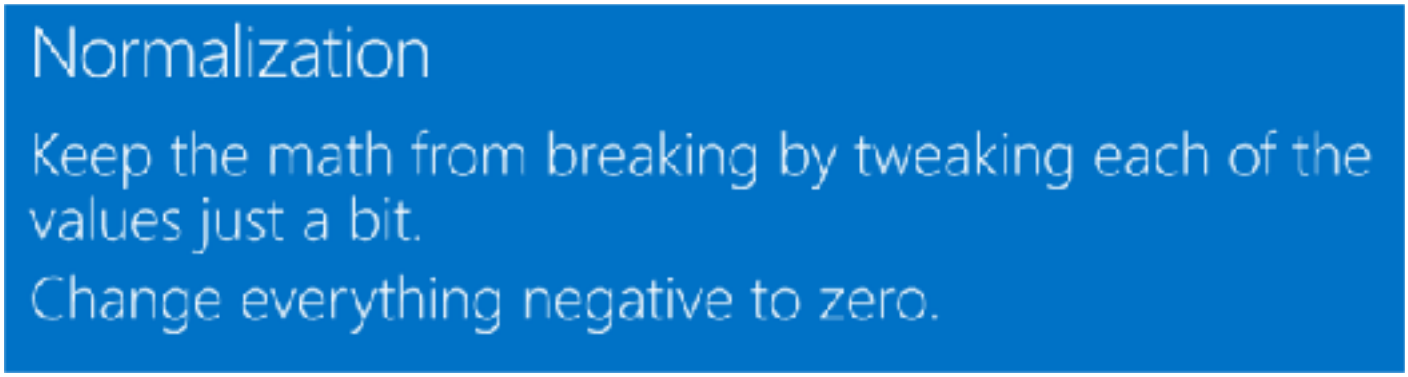
因为最大池化（max-pooling）保留了每一个小块内的最大值，所以它相当于保留了这一块最佳的匹配结果（因为值越接近1表示匹配越好）。这也就意味着它不会具体关注窗口内到底是哪一个地方匹配了，而只关注是不是有某个地方匹配上了。这也就能够看出，CNN能够发现图像中是否具有某种特征，而不用在意到底在哪里具有这种特征。这也就能够帮助解决之前提到的计算机逐一像素匹配的死板做法。

当对所有的feature map执行池化操作之后，相当于一系列输入的大图变成了一系列小图。同样地，我们可以将这整个操作看作是一个操作，这也就是CNN中的池化层(pooling layer)，如下：



通过加入池化层，可以很大程度上减少计算量，降低机器负载。

Normalization

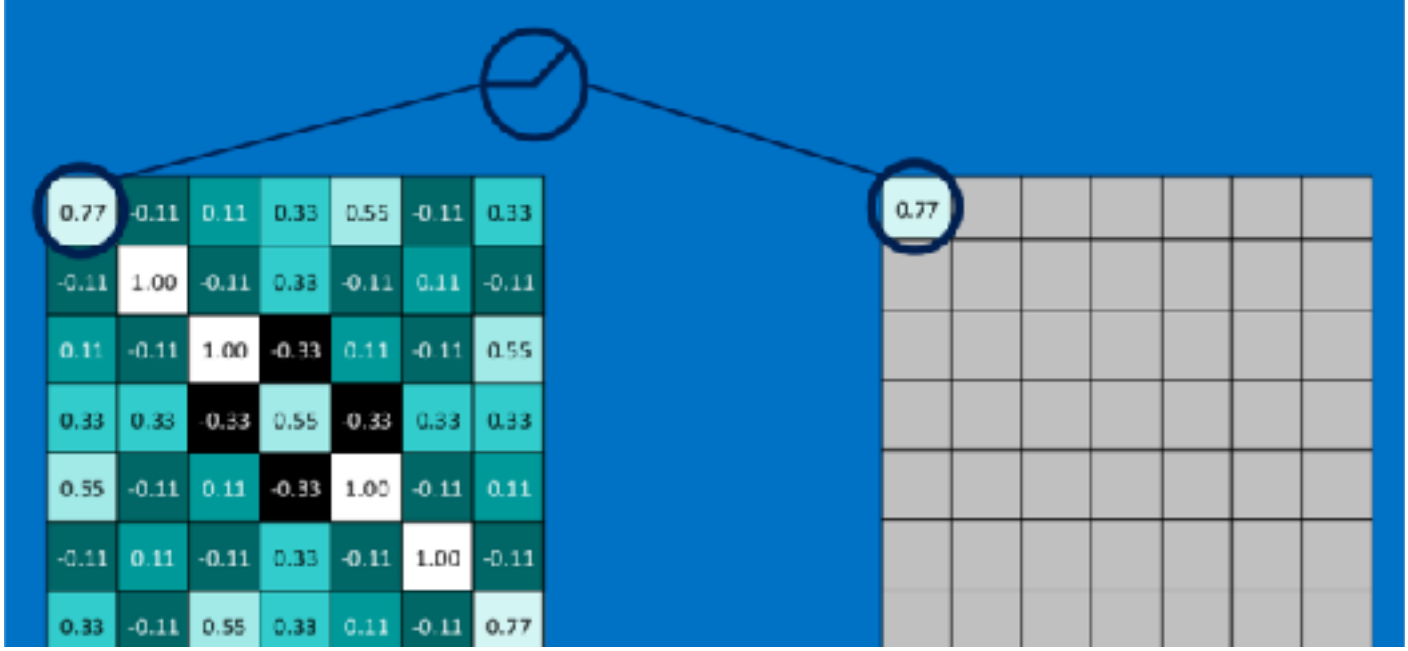


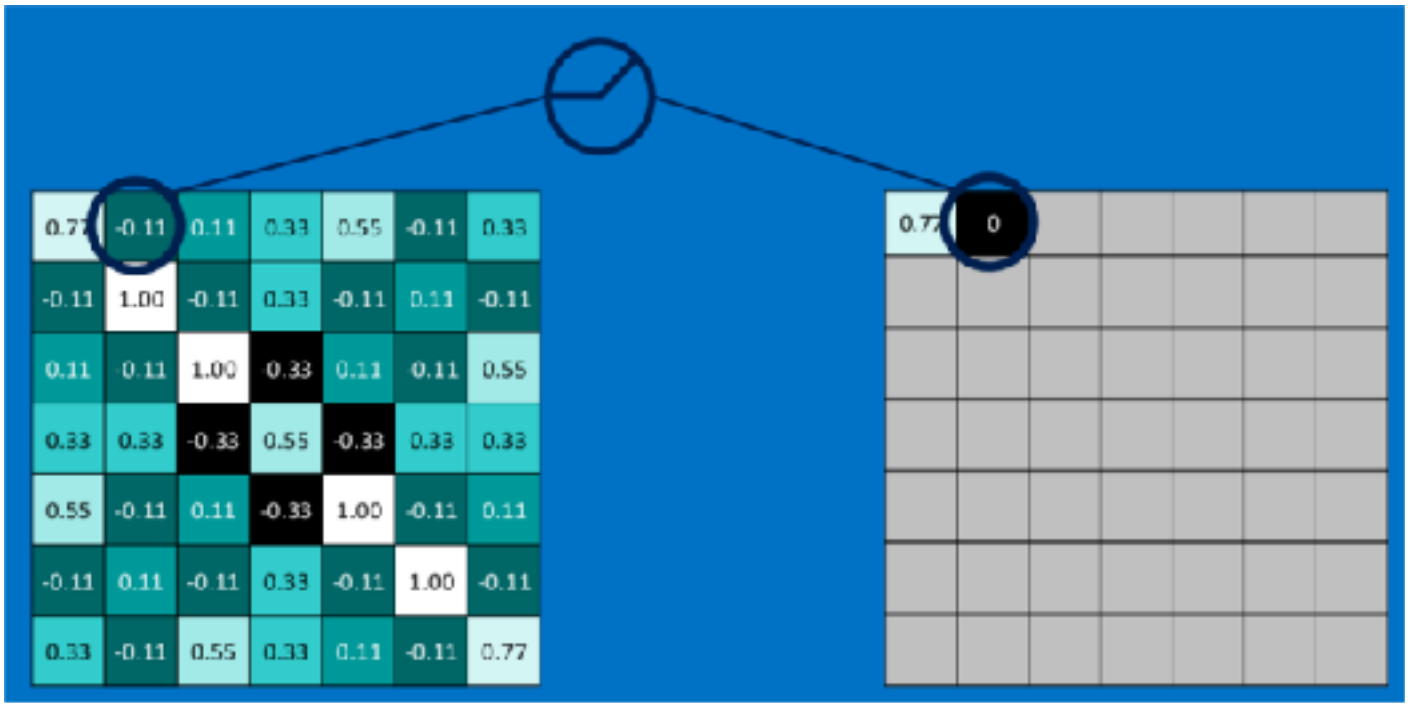
激活函数Relu (Rectified Linear Units)

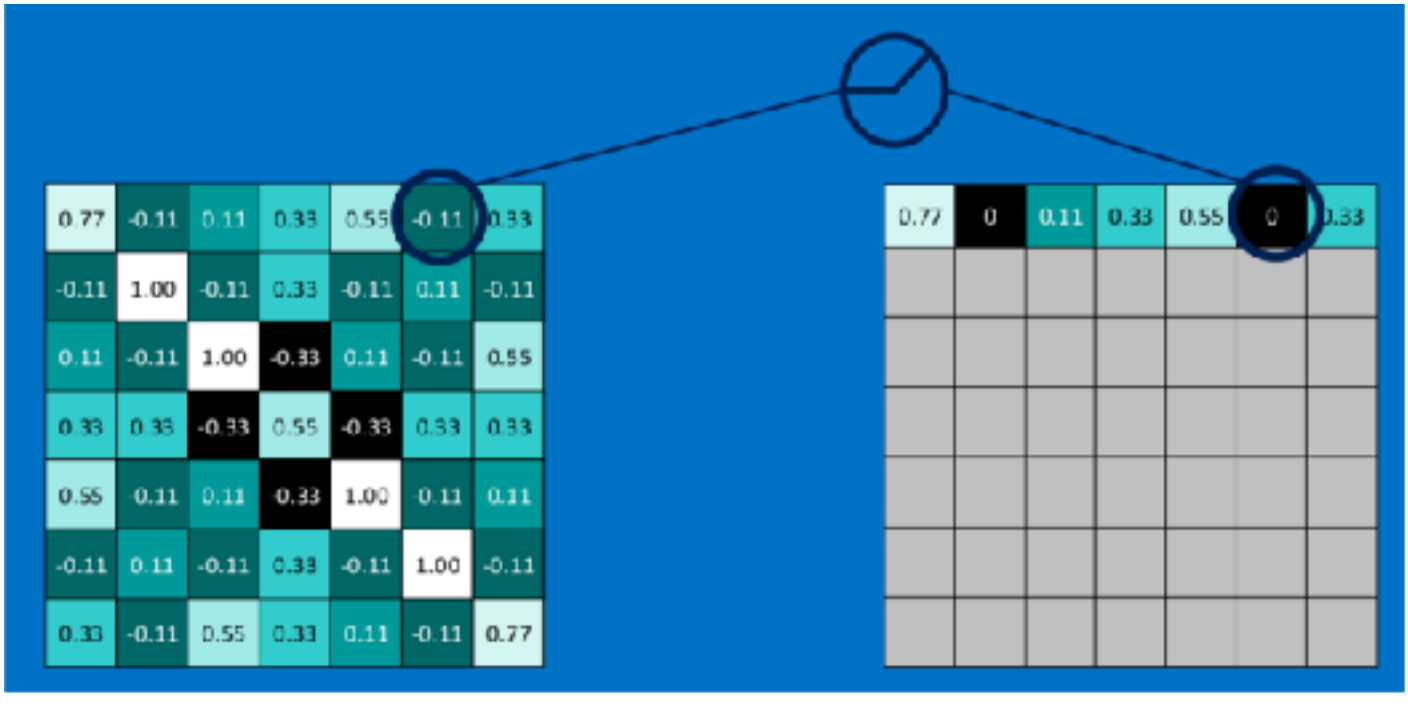
这是一个很小但是很重要的操作，叫做Relu(Rectified Linear Units)，或者修正线性单元。它的数学公式也很简单:
![][01]
[01]:http://latex.codecogs.com/png.latex?f(x)%20=%20max(0,%20x)
(https://link.jianshu.com?t=http://latex.codecogs.com/png.latex?f(x)%20=%20max(0,%20x))

对于输入的负值，输出全为0，对于正值，原样输出。关于其功能，更多详见 这里(https://link.jianshu.com?t=http://cs231n.github.io/neural-networks-1/)。

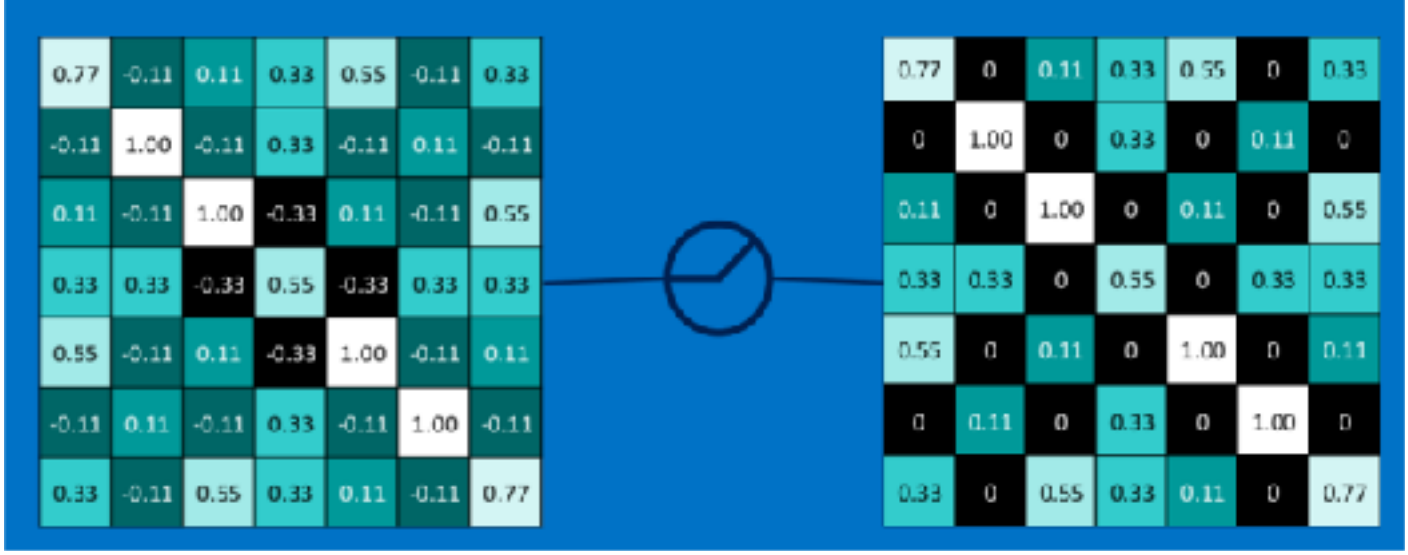
下面我们看一下本文的离例子中relu激活函数具体操作：



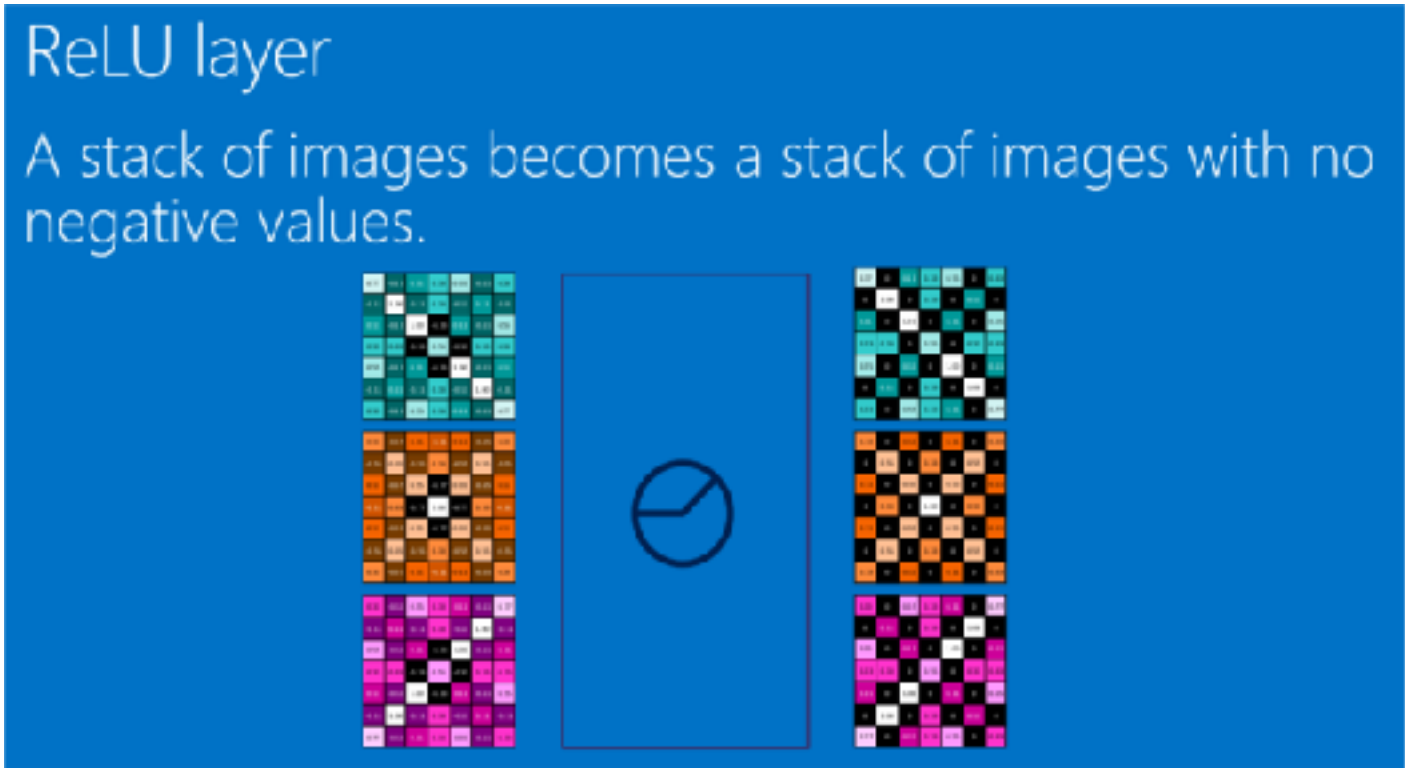




最后，对整幅图操作之后，结果如下：



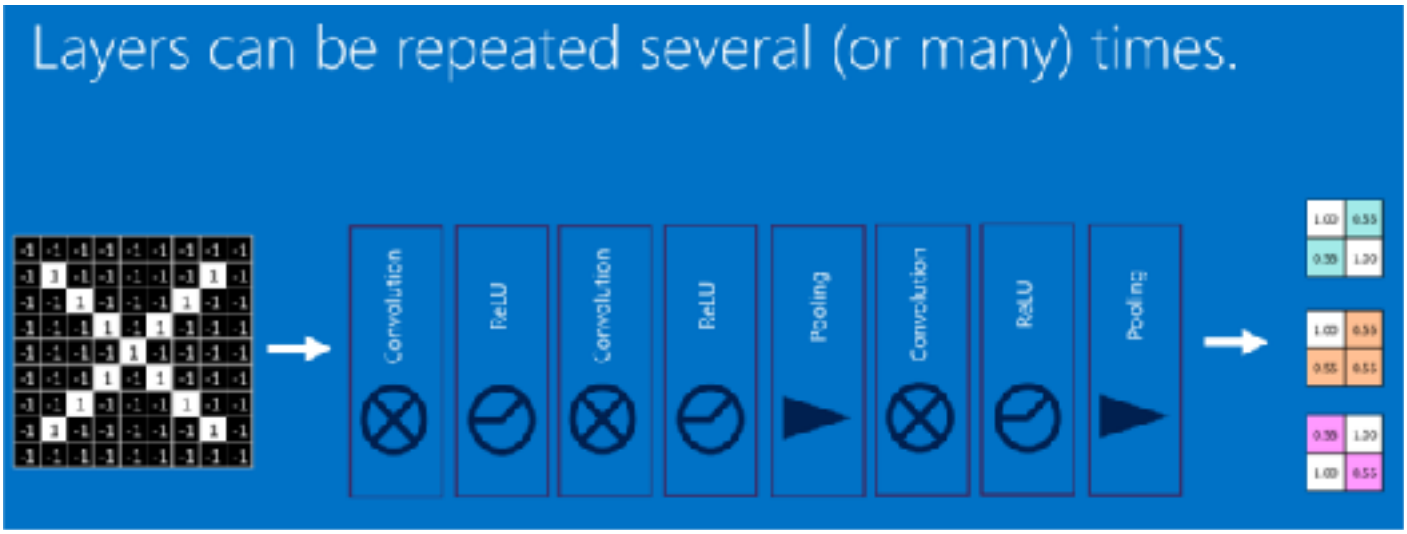
同样地，在CNN中，我们这一系列操作视为一个操作，那么就得到Relu Layer，如下：



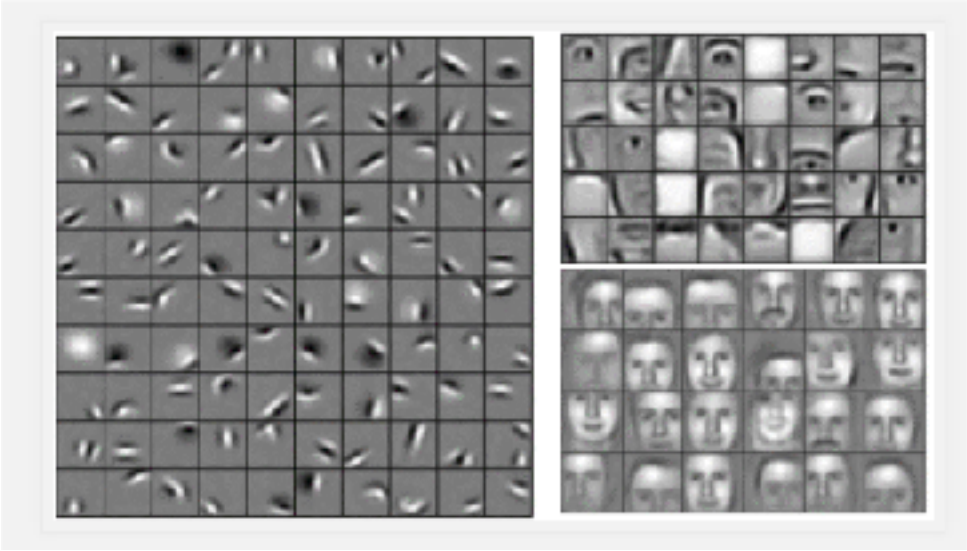
Deep Learning

最后，我们将上面所提到的卷积，池化，激活放在一起，就是下面这个样子：

然后，我们加大网络的深度，增加更多的层，就得到深度神经网络了：



然后在不同的层，我们进行可视化，就可以看到本文开头提到的先验知识里面的结果了：

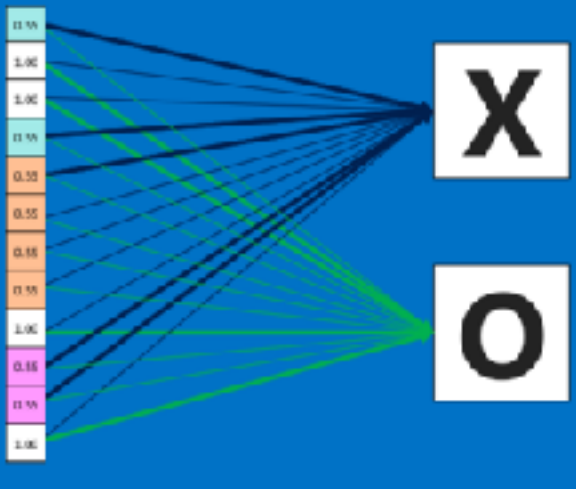


全连接层(Fully connected layers)

Every value gets a vote

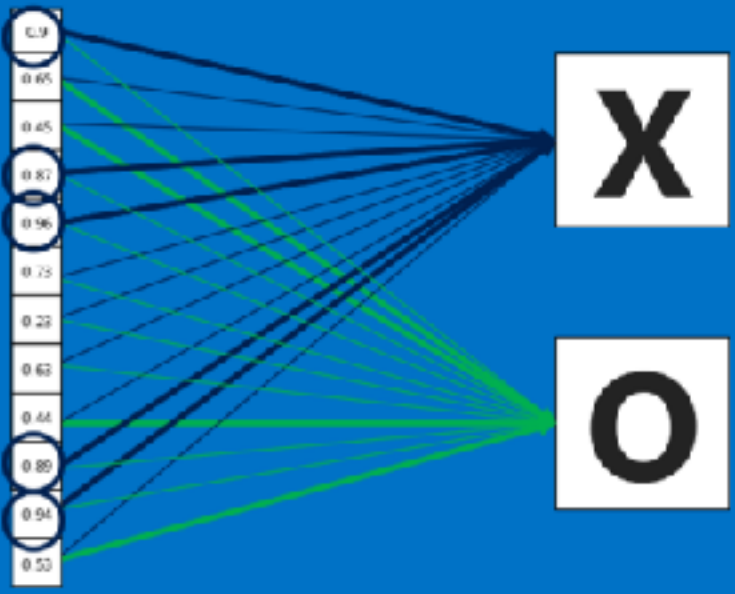
Vote depends on how strongly a value predicts X or O

Vote depends on how strongly a value predicts X or O

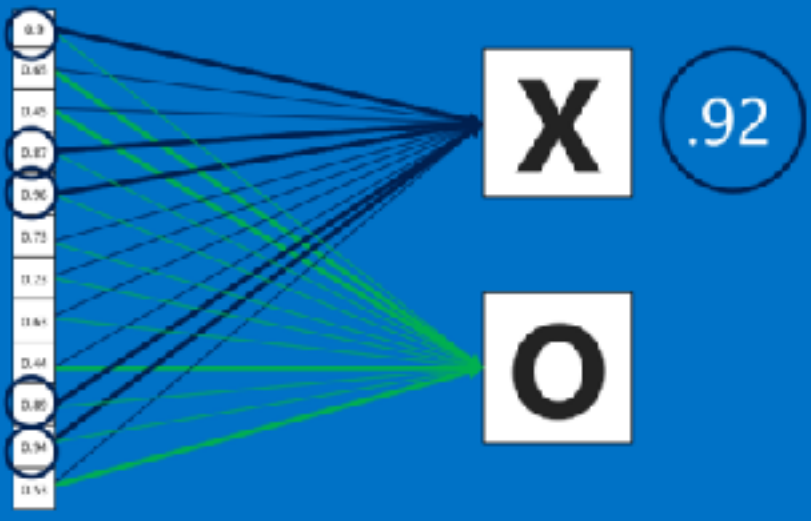


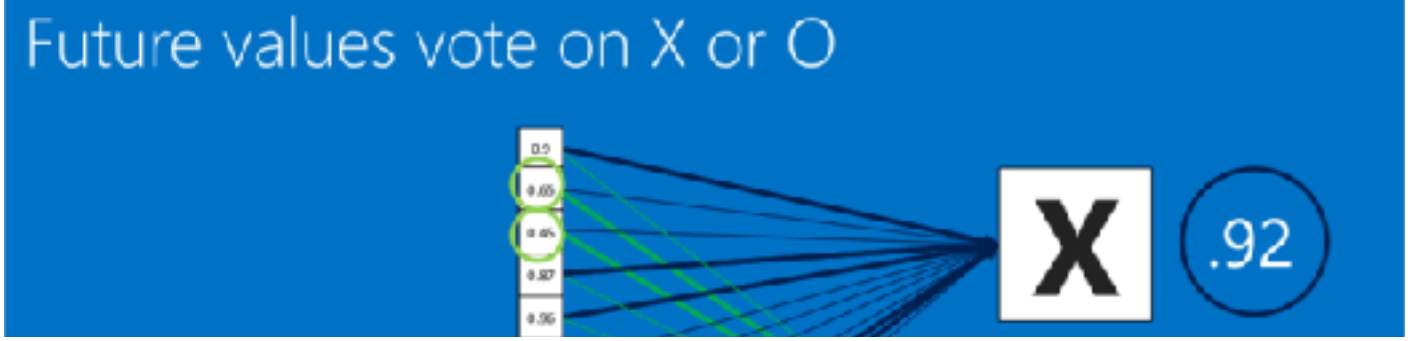
Future values vote on X or O

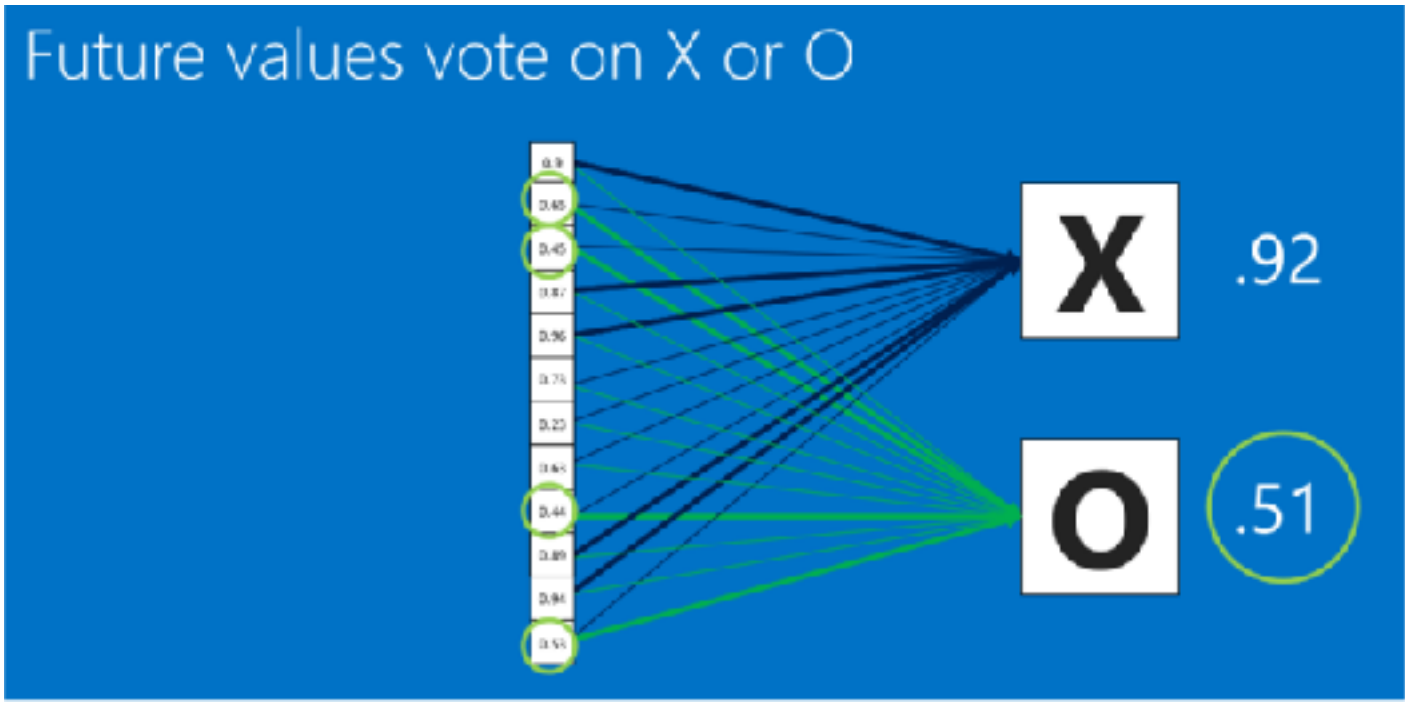
Future values vote on X or O



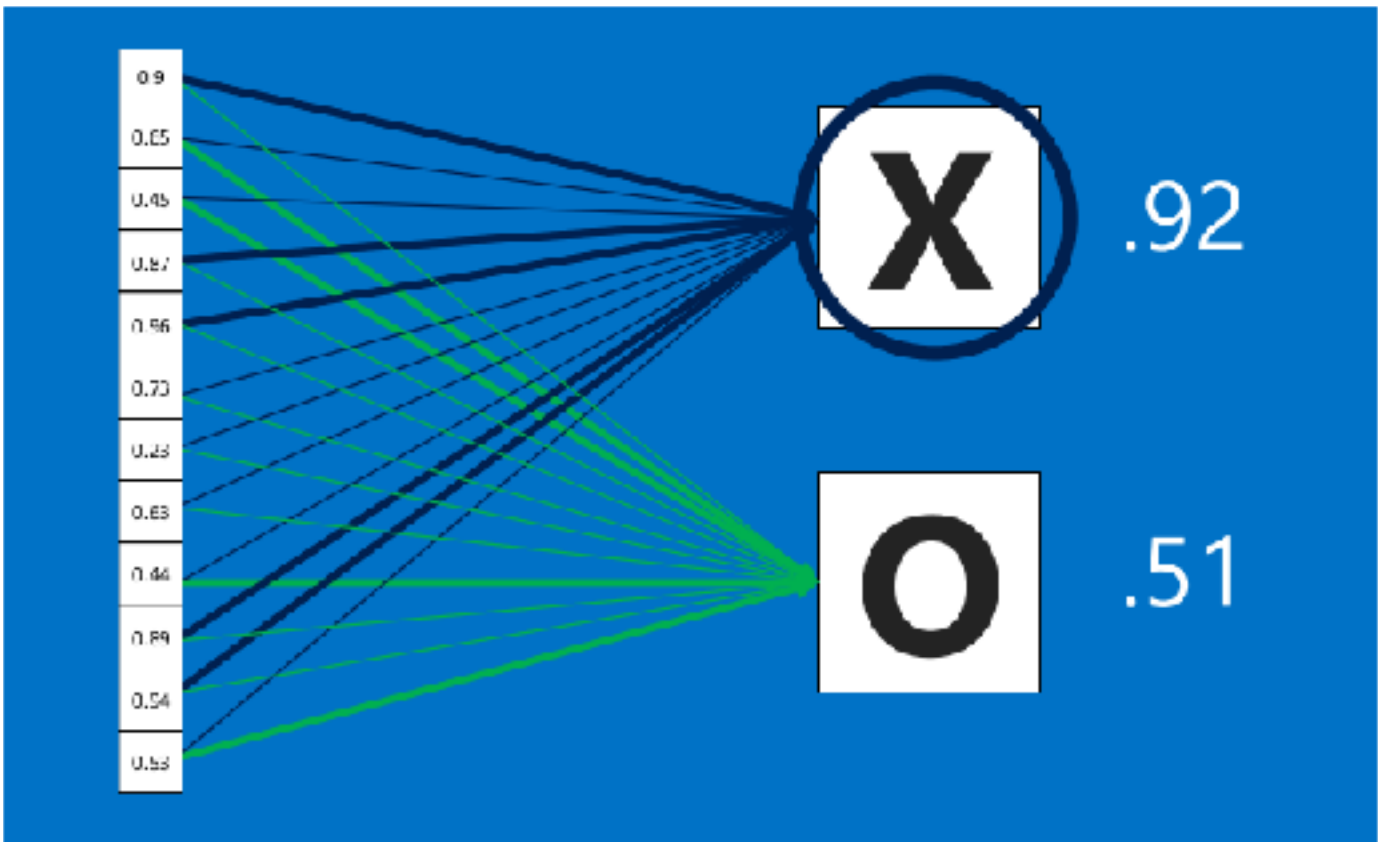
Future values vote on X or O



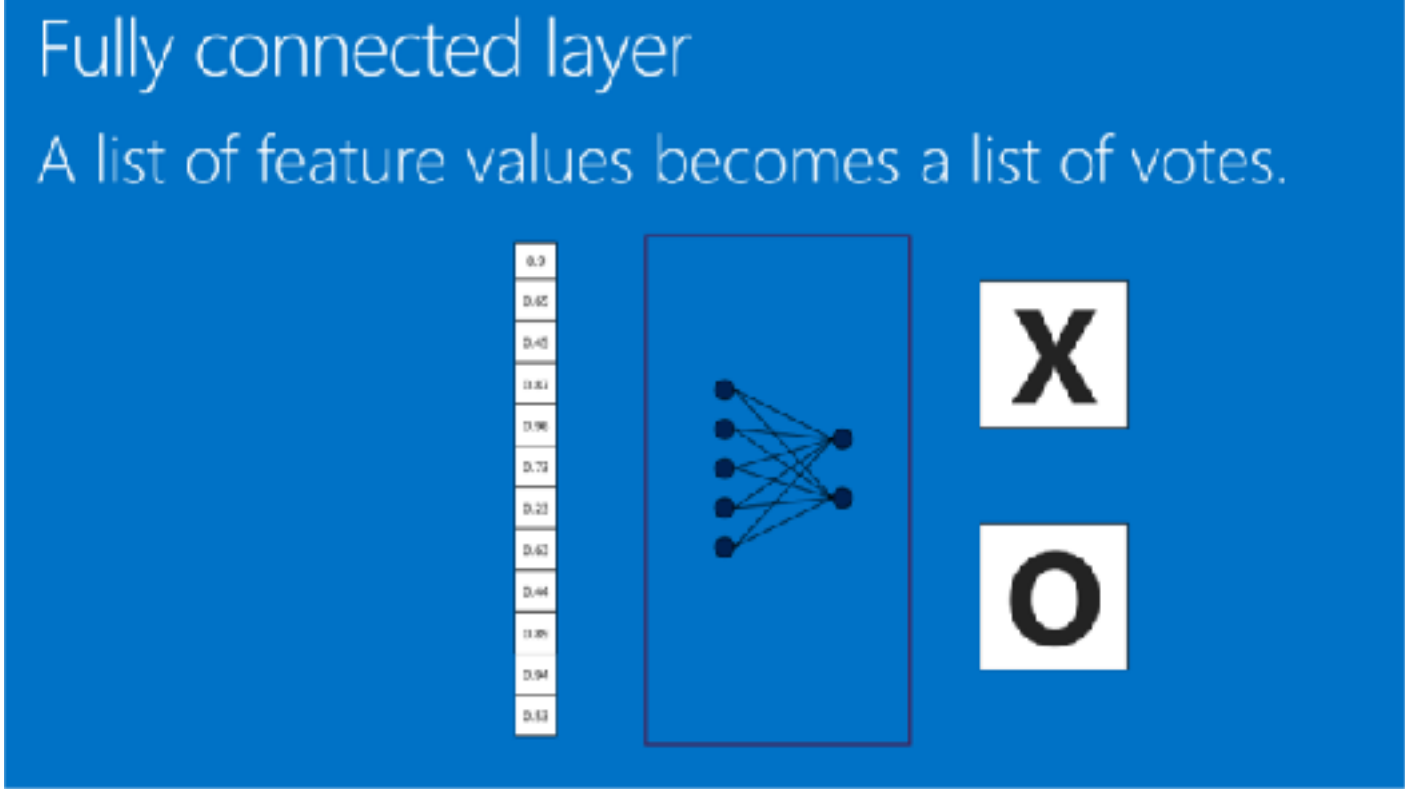




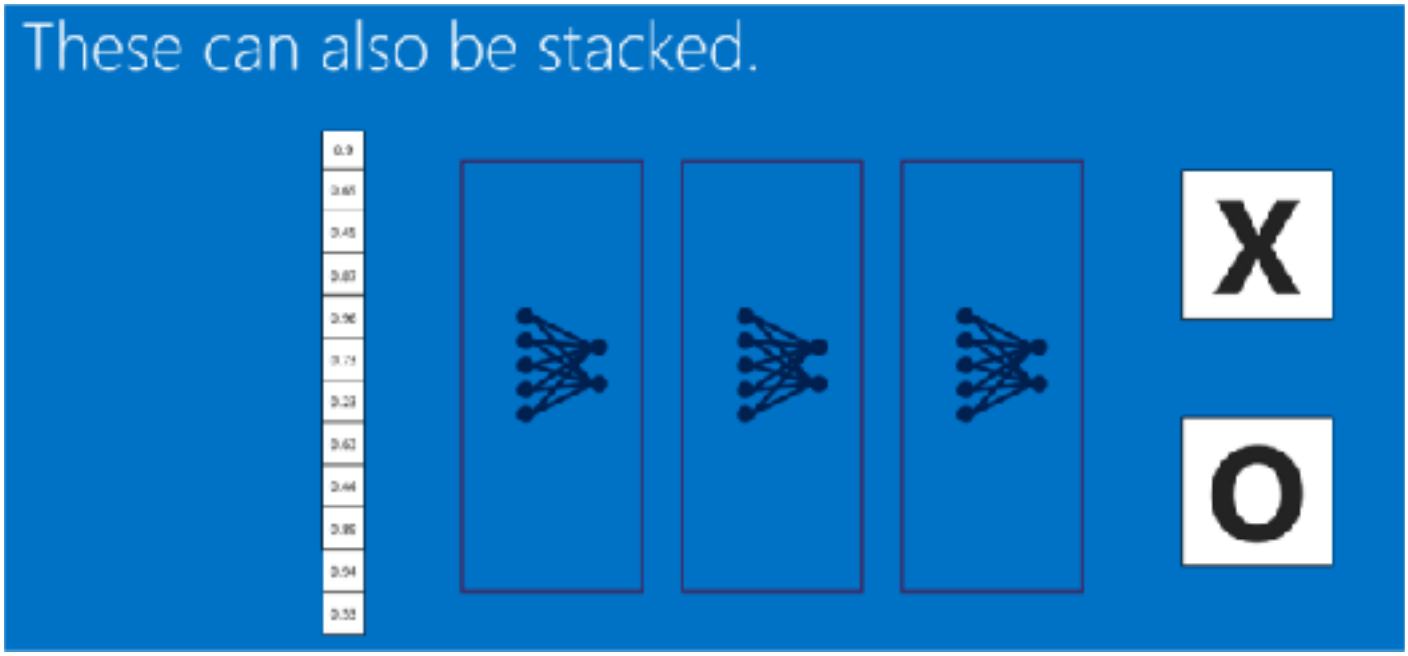
根据结果判定为"X"：



在这个过程中，我们定义这一系列操作为”全连接层“(Fully connected layers)：



全连接层也能够有很多个，如下：

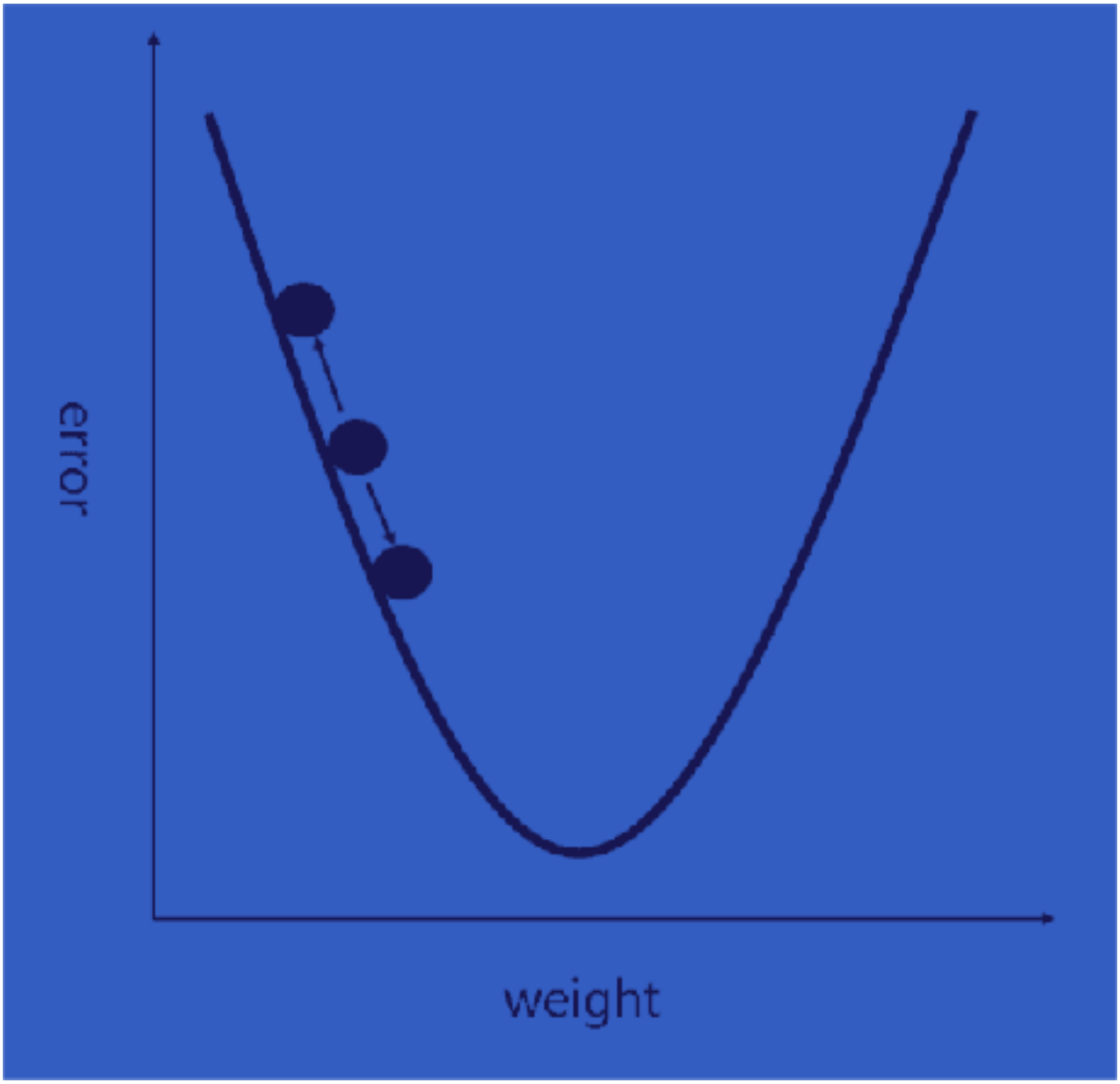


【综合上述所有结构】

这一整个过程，从前到后，被称作”前向传播“，得到一组输出，然后通过反向传播来不断纠正错误，进行学习。

反向传播 (Backpropagation)

此处数学原理可以参见：深度学习 —— 反向传播理论推导 (https://www.jianshu.com/p/408ab8177a53).



Learning

Q: Where do all the magic numbers come from ?

Features in convolutional layers

Backprop

	Right answer	Actual answer	Error
X	1		
O			

Backprop

	Right answer	Actual answer	Error
X	1	0.92	
O			

Backprop

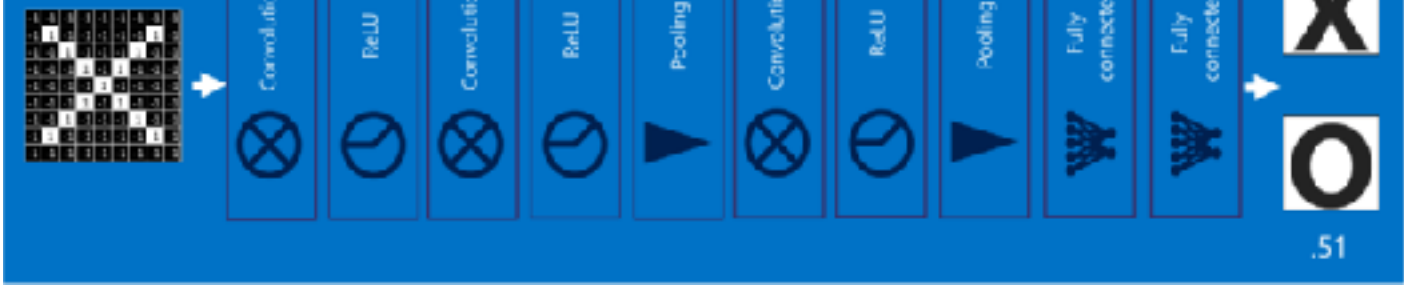
	Right answer	Actual answer	Error
X	1	0.92	0.08

Backprop

	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49

Backprop

	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49
		Total	0.57

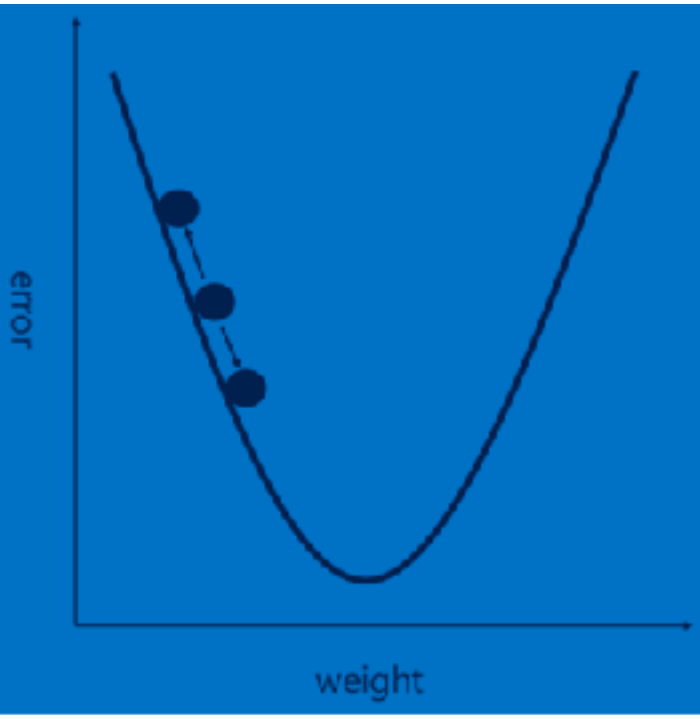


Gradient Descent Optimizer

Gradient descent

Gradient descent

For each feature pixel and voting weight, adjust it up and down a bit and see how the error changes.

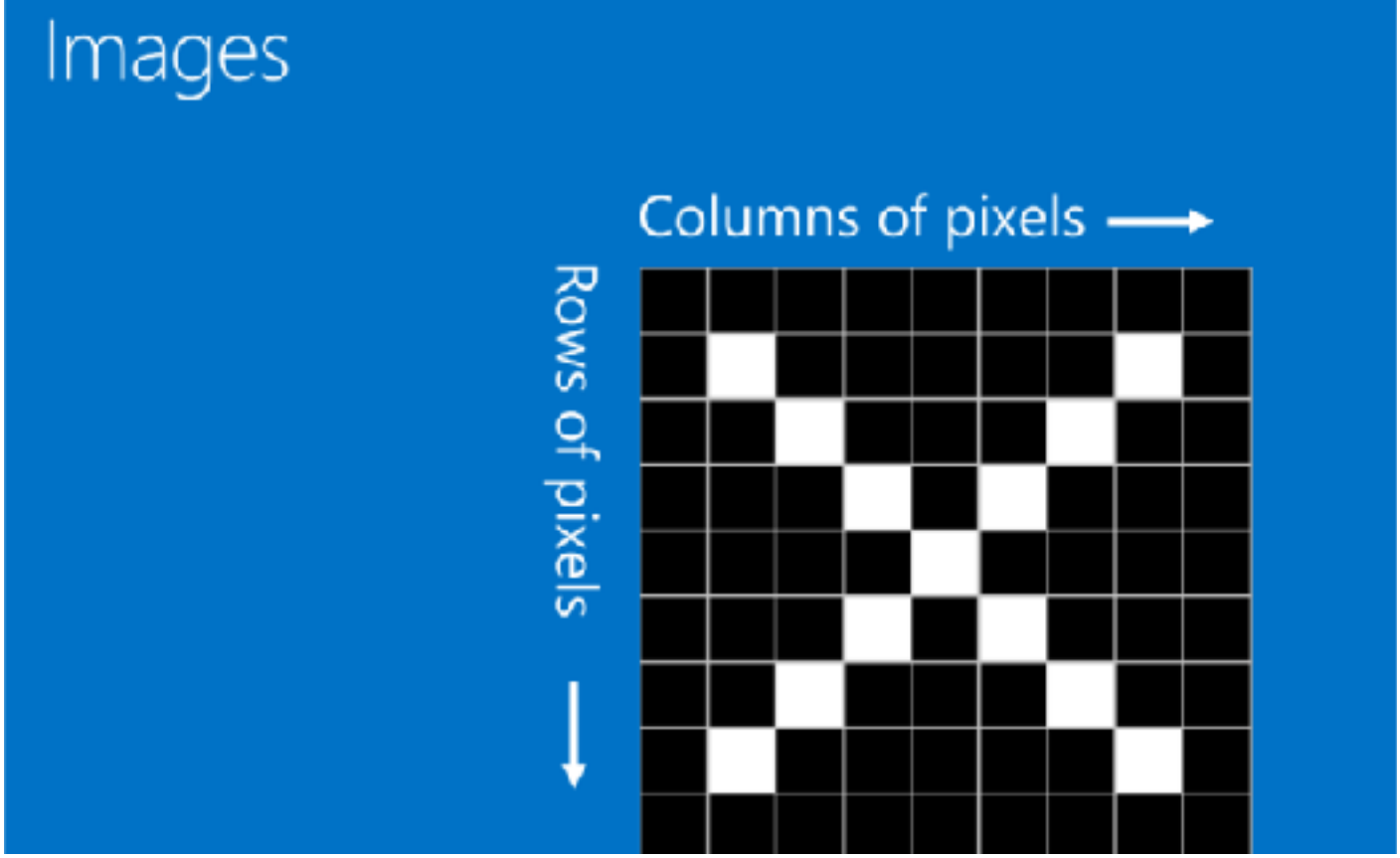


Hyperparameters

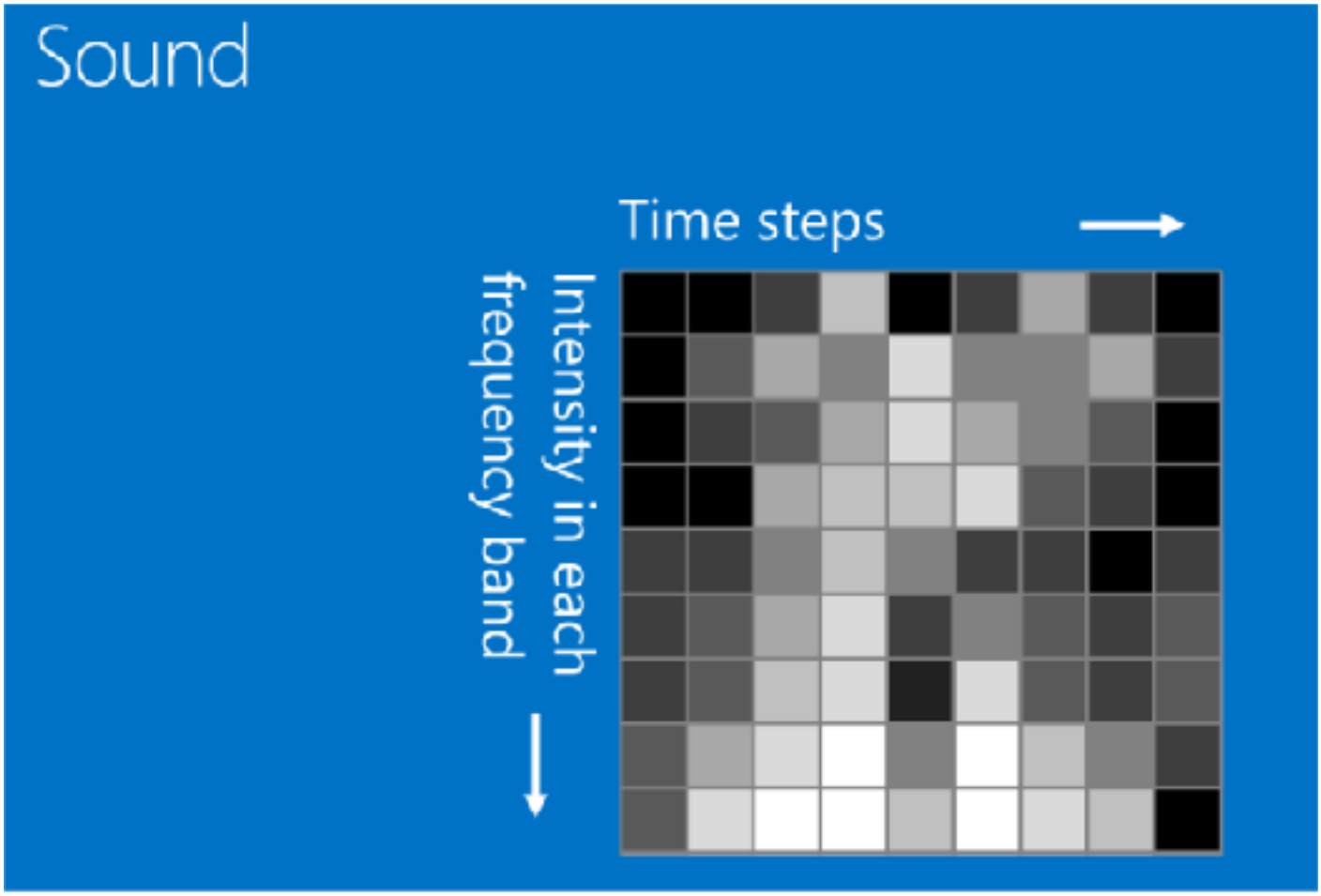
Hyperparameters (knobs)

- Convolution
  - Number of features
  - Size of features
- Pooling
  - Window size
  - Window stride
- Fully Connected
  - Number of neurons

Application



Images

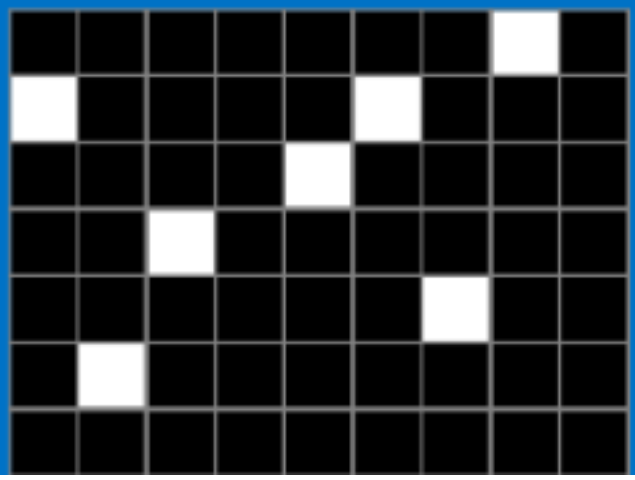


Soung

Text

Position in sentence →

Words in dictionary ↓



Text

Limitations

ConvNets only capture local "spatial" patterns in data.

If the data can't be made to look like an image, ConvNets are less useful.

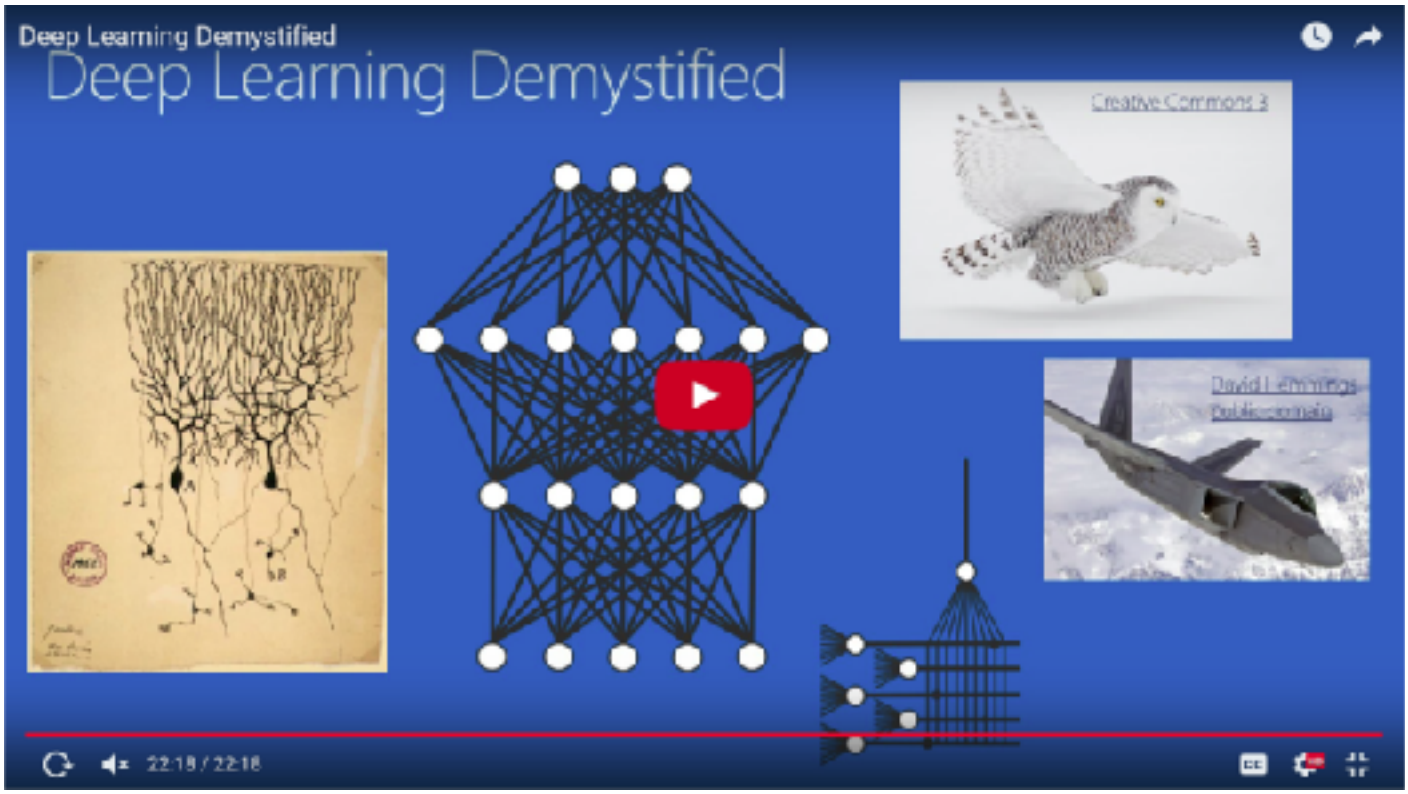
Customer data

Name, age, address, email, purchases, browsing activity,... →

Customers ↓

A	22	1A	a@a	1	aa	a1.a	123	aa1
B	33	2B	b@b	2	bb	b2.b	234	bb2
C	44	3C	c@c	3	cc	c3.c	345	cc3
D	55	4D	d@d	4	dd	d4.d	456	dd4
E	66	5E	e@e	5	ee	e5.e	567	ee5
F	77	6F	f@f	6	ff	f6.f	678	ff6
G	88	7G	g@g	7	gg	g7.g	789	gg7
H	99	8H	h@h	8	hh	h8.h	890	hh8
I	111	9I	i@i	9	ii	i9.i	901	ii9

Learn more



If you'd like to dig deeper into deep learning, check out my Demystifying Deep Learning post (https://link.jianshu.com?t=http://brohrer.github.io/deep_learning_demystified.html). I also recommend the notes from the Stanford CS 231 course (https://link.jianshu.com?t=http://cs231n.github.io/convolutional-networks/) by Justin Johnson and Andrej Karpathy that provided inspiration for this post, as well as the writings of Christopher Olah (https://link.jianshu.com?t=http://colah.github.io/archive.html), an exceptionally clear writer on the subject of neural networks.

If you are one who loves to learn by doing, there are a number of popular deep learning tools available. Try them all! And then tell us what you think.
Caffe (https://link.jianshu.com?t=http://caffe.berkeleyvision.org/)
CNTK (https://link.jianshu.com?t=https://github.com/Microsoft/CNTK)
Deeplearning4j (https://link.jianshu.com?t=https://en.wikipedia.org/wiki/Deeplearning4j)
TensorFlow (https://link.jianshu.com?t=http://www.tensorflow.org/)
Theano (https://link.jianshu.com?t=https://en.wikipedia.org/wiki/Theano_(software))
Torch (https://link.jianshu.com?t=https://en.wikipedia.org/wiki/Torch_(machine_learning))
Many others (https://link.jianshu.com?t=http://deeplearning.net/software_links/)

I hope you've enjoyed our walk through the neighborhood of Convolutional Neural Networks. Feel free to start up a conversation.

Brandon Rohrer

本文重点在前面的具体操作部分，所以后面的部分没有细说，只是带过了，如有必要，后期会逐渐完善，谢谢！

Reference：

- http://brohrer.github.io/how_convolutional_neural_networks_work.html (https://link.jianshu.com?t=http://brohrer.github.io/how_convolutional_neural_networks_work.html)
- 深度学习 —— 反向传播理论推导 (https://www.jianshu.com/p/408ab8177a53)

(注：感谢您的阅读，希望本文对您有所帮助。如果觉得不错欢迎分享转载，但请先点击这里 (https://link.jianshu.com?t=https://101705160006292.bqy.mobi/) 获取授权。本文由版权印 (https://link.jianshu.com?t=https://www.banquanyin.com/) 提供保护，禁止任何形式的未授权违规转载，谢谢！)